TEXAS INSTRUMENTS

MSC121x
Precision ADC and DACs with 8051 Microcontroller and Flash Memory

User's Guide

April 2005 *Data Acquisition Products*

SBAU101

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Copyright © 2005, Texas Instruments Incorporated

Read This First

About This Manual

This users guide describes the function and operation of the MSC121x precision ADC and DACs with 8051 microcontroller and flash memory.

Information About Cautions and Warnings

This book may contain cautions and warnings.

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

This is an example of a warning statement.

A warning statement describes a situation that could potentially cause harm to you.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

Related Documentation and Tools From Texas Instruments

Data Sheet:	Literature Number:
MSC1210	SBAS203
MSC1211	SBAS267
MSC1212	SBAS278
MSC1213	SBAS323
MSC1214	SBAS323
User's Guide:	**Literature Number:**
MSC1211EVM User's Guide	SLAU086
MSC1210EVM User's Guide	
MSC1210−DAQ−EVM User's Guide	

For a complete list of application notes and related documentation, see the MSC web site at www.ti.com/msc.

Trademarks

I^2C is a trademark of Koninklijke Philips Electronics N.V.

SPI is a trademark of Motorola.

All other trademarks are the property of their respective owners.

Contents

1 Introduction 1-1
1.1 MSC121x Description 1-2
1.2 MSC121x Pinout 1-4
1.2.1 Input/Output (I/O) Ports − P0, P1, P2, and P3 1-7
1.2.2 Oscillator XOUT (pin 1) and XIN (pin 2) 1-10
1.2.3 Reset Line − RST (pin 13) 1-10
1.2.4 Address Latch Enable − ALE (pin 45) 1-10
1.2.5 Program Store Enable − $\overline{\text{PSEN}}$ (pin 44) 1-11
1.2.6 External Access − $\overline{\text{EA}}$ (pin 48) 1-11
1.3 Enhanced 8051 Core 1-12
1.4 Family Compatibility 1-13
1.5 Flash Memory 1-13
1.6 Internal SRAM 1-13
1.7 High-Performance Analog Functions 1-13
1.8 High-Performance Peripherals 1-14

2 MSC121x Addressable Resources 2-1
2.1 Introduction 2-2
2.2 Program Memory and Data Memory 2-3
2.3 Core Data Memory and Special Function Registers 2-5
2.4 Beyond 64 KBytes 2-6

3 Special Functions Registers 3-1
3.1 Introduction 3-2
3.2 Referencing SFRs in Assembly and C Languages 3-3
3.3 SFR Types 3-4
3.4 SFR Overview 3-4

4 Programmer's Model and Instruction Set 4-1
4.1 Introduction 4-2
4.2 Registers 4-2
4.3 Instruction Types and Addressing Modes 4-4
4.4 MSC121x Op-Code Table 4-10
4.5 Examples of MSC121x Instructions 4-12

5 System Clocks, Timers, and Functions 5-1
5.1 Timing Chain and Clock Controls 5-2
5.2 System Clock Divider (MSC1211/12/13/14) 5-5
5.2.1 Behavior in Delay Mode 10_2 5-6
5.3 Watchdog Timer 5-6
5.3.1 Watchdog Timer Example Program 5-7
5.4 Low-Voltage Detection 5-8
5.5 Hardware Configuration 5-9
5.6 Breakpoints 5-11

6 Analog-To-Digital Converters 6-1
6.1 ADC Functional Blocks 6-2
6.2 ADC Signal Flow and General Description 6-3
6.3 Analog Input Stage 6-3
6.4 Input Impedance, PGA, and Voltage References 6-5
6.5 Offset DAC 6-8
6.6 ADC Data Rate, Filters, and Calibration 6-9
6.7 32-Bit Summation Register 6-13
6.8 Accessing the ADC Multi-Byte Conversion in C 6-15
6.9 ADC Example Program 6-17

7 Digital-To-Analog Converters 7-1
7.1 Introduction 7-2
7.2 DAC Selection 7-3
7.3 DAC Configuration and Control 7-5
7.4 DAC Technology and Limitations 7-7
7.5 DAC Example Program 7-8

8 Pulse-Width Modulator and Tone Generator 8-1
8.1 Description 8-2
8.2 PWM Generator Example 8-4

9 Inter-IC (I^2C™) Subsystem 9-1
9.1 Introduction to the I^2C Bus 9-2
9.2 I^2C Terminology 9-2
9.3 I^2C Bus Lines and Basic Timing 9-3
9.4 I^2C Data Transfers and the Acknowledge Bit 9-4
9.5 I^2C Principal Registers 9-6
9.6 I^2C Related Registers 9-10
9.7 I^2C Example—MSC1211/13 as a Master 9-11
9.8 I^2C Example—MSC1211/13 as a Slave 9-13
9.9 I^2C Example—MSC1211/13 as an Interrupt-Driven Slave 9-15
9.10 I^2C Synchronization and Arbitration 9-17
9.11 I^2C Fast Mode 9-17
9.12 I^2C General Call 9-17
9.13 I^2C 10-Bit Addressing 9-18

10 Serial Peripheral Interface (SPI™) 10-1
10.1 Description 10-2
10.2 SPI Configuration 10-2
10.3 SPI Interrupts 10-5
10.4 SPI FIFO Buffer 10-6
10.5 SPI Examples 10-9

11 Timers and Counters 11-1
11.1 Description 11-2
11.2 Timer/Counters 0 and 1 11-3
11.2.1 Modes 0 and 1 11-5
11.2.2 Mode 2 11-6
11.2.3 Mode 3 11-7
11.2.4 Summary of Control Bits and SFRs for Timer/Counters 0 and 1 11-7
11.3 Timer/Counter 2 11-8
11.3.1 16-Bit Timer/Counter with Optional Capture 11-9
11.3.2 16-Bit Timer/Counter with Automatic and Forced Reload 11-10
11.3.3 Baud Rate Generator 11-11
11.3.4 Summary of Control Bits and SFRs for Timer/Counter 2 11-12
11.4 Example Program Using Timers 0, 1, and 2 11-13

12 Serial Ports (USART0 and USART1) 12-1
12.1 Description 12-2
12.2 Control Bits in SCON0 and SCON1 12-3
12.3 Pin and Interrupt Assignments 12-4
12.4 Timer/Counters 1 and 2 Baud Rate Generation 12-5
12.5 Mode 0—8-Bit Synchronous 12-7
12.6 Mode 1—10-Bit Asynchronous 12-8
12.7 Modes 2 and 3—11-Bit Asynchronous 12-9
12.8 Multiprocessor Communications 12-10
12.9 Example Program 12-10

13 Interrupts 13-1
13.1 Description 13-2
13.2 Standard and Extended Interrupts 13-2
13.3 Auxiliary Interrupt Sources 13-4
13.4 Multiple Interrupts 13-5
13.5 Example of Multiple and Nested Interrupts 13-6
13.6 Example of Wake Up from Idle 13-9

Figures

1−1. MSC121x Block Diagram 1-2
1−2. MSC121x Pin Configuration 1-4
1−3. Standard 8051 I/O Pin Structure 1-7
1−4. CMOS Output Pin Structure 1-7
1−4. Open-Drain Output Pin Structure 1-7
1−6. Input Pin Structure 1-8
1−7. Comparison of MSC121x Timing to Standard 8051 Timing 1-12
2−1. On-Chip and Off-Chip Resources 2-2
5−1. MSC121x Timing Chain and Clock Control 5-2
6−1. ADC Subsystem Elements 6-2
6−2. Input Multiplexer Configuration 6-4
6−3. Analog Input Structure without Buffer 6-5
6−4. Filter Frequency Responses 6-10
7−1. DAC Architecture 7-2
9−1. I^2C Bus Connection of Standard and Fast Mode Devices 9-3
9−2. START and STOP Conditions 9-3
9−3. I^2C-Bus Bit Transfer 9-3
9−4. I^2C-Bus Data Transfer 9-4
9−5. I^2C Acknowledge 9-5
10−1. SPI Master/Slave Interconnect 10-2
10−2. SPI Clock/Data Timing 10-4
10−3. SPI FIFO Operation 10-6
11−1. Timer 0/1—Modes 0 and 1 11-5
11−2. Timer 0/1—Mode 2 11-6
11−3. Timer 0—Mode 3 11-7
11−4. Timer/Counter 2—16-Bit with Capture 11-9
11−5. Timer/Counter 2—16-Bit with Reload 11-10
11−6. Timer/Counter 2—Baud Rate Generator 11-11
12−1. Synchronous Receive at $f_{CLK}/4$ 12-7
12−2. Synchronous Transmit at $f_{CLK}/4$ 12-7
12−3. Asynchronous 10-Bit Transmit Timing 12-8
12−4. Asynchronous 10-Bit Receive Timing 12-8
12−5. Asynchronous 11-Bit Receive 12-9
12−6. Asynchronous 11-Bit Transmit 12-9
12−7. Serial Port with Software Buffer 12-10

Tables

1−1. MSC121x Product Family Matrix 1-3
1−2. MSC121x Pin Descriptions 1-5
1−3. Port 1 Alternate Functions 1-8
1−4. Port 3 Alternate Functions 1-9
2−1. Program Memory and External Data Memory Addresses 2-3
2−2. MSC121x Flash Memory Partitioning and Addresses 2-4
2−3. On-Chip 8051 Memory 2-5
3−1. Special Function Register Map 3-2
3−2. SFR Overview 3-4
4−1. 8051 Working Registers 4-2
4−2. Symbol Descriptions for Instruction List (Table 4−3) 4-5
4−3. Instruction List 4-5
4−4. MSC121x Op-Codes 4-10
5−1. SYSCLK—System Clock Divider Register 5-5
5−2. Watchdog Control Bits 5-6
5−3. LVDCON—Low Voltage Detect Control 5-8
5−4. Low Voltage Detect 5-8
5−5. Hardware Configuration Register 0 5-9
5−6. Hardware Configuration Register 1 5-10
5−7. MCON—Memory Control 5-11
5−8. BPCON—Breakpoint Control 5-11
5−9. BPL—Breakpoint Low Address for BP Register Selected in MCON at 95h 5-12
5−10. BPH—Breakpoint High Address for BP Register Selected in MCON at 95h 5-12
5−11. Breakpoints 5-12
6−1. ADMUX—ADC Multiplexer 6-4
6−2. Impedance Divisor (G) for a Given PGA 6-5
6−3. ADCON0—ADC Control Register 0 6-6
6−4. ADCON0 Bit Parameters 6-7
6−5. ADCON1—ADC Control Register 1 6-9
6−6. ADC Interrupt Controls 6-11
6−7. Summation Register 6-13
6−8. SSCON—Summation/Shift Control 6-13
6−9. Summation Interrupt Controls 6-14
7−1. DACSEL Values 7-3
7−2. LOADCON SFR 7-3
7−3. DxLOAD Output Modes for DACx 7-3
7−4. DAC Control Registers 7-5
8−1. PWMCON—PWM Control 8-2
8−2. PWM Output 8-3
9−1. I^2C Terminology 9-2
9−2. I2CCON—I^2C Control Register 9-6
9−3. I2CDATA SFR 9-7
9−4. I2CGM—I^2C General Call / Multiple Master Control 9-7
9−5. I2CSTAT SFR 9-8
9−6. I^2C Status Codes 9-8
9−7. PDCON of I^2C and SPI 9-10

9−8. Interrupt Control for I2C 9-10
9−9. Address Allocation 9-18
10−1. SPICON—SPI Control 10-3
10−2. P1—Port 1 10-3
10−3. P1DDRH—Port 1 Data Direction Register 10-4
10−4. SPIDATA—SPI Data Register 10-4
10−5. SPI Interrupts Have Highest Priority and Jump to Address 0033h 10-5
10−6. PAI—Pending Auxiliary Interrupt Register 10-5
10−7. SPISTART—SPI Buffer Start Address 10-7
10−8. SPISEND—SPI Buffer End Address 10-7
10−9. SPIRCON—SPI Receive Control Register 10-8
10−10. SPITCON—SPI Transmit Control Register 10-8
11−1. TMOD—Timer Mode Control 11-3
11−2. TCON—Timer/Counter Control 11-4
11−3. Modes 0 and 1 Operation 11-5
11−4. Control Bit and SFR Summary for Timer/Counters 0 and 1 11-7
11−5. T2CON—Timer 2 Control 11-8
11−6. Control Bit and SFR Summary for Timer/Counter 2 11-12
11−7. Timer Modes 11-13
12−1. SCON0 and SCON1—Serial Port 0 and Serial Port 1 Control 12-3
12−2. USART Pin and Interrupt Assignments 12-4
12−3. Timer/Counter 2 Baud Rate Generation 12-5
12−4. Timer/Counter 1 Baud Rate Generation 12-5
12−5. USART Baud Rate Generation 12-6
13−1. Standard and Extended Interrupts 13-3
13−2. Auxiliary Interrupts with Highest Group Priority 13-4
13−3. EWU—Enable Wake Up 13-9

Examples

2−1. Instructions 2-6
3−1. Assembly Code and C Code 3-3
4−1. Intruction Types 4-4
4−2. Instruction Addressing Modes 4-4
4−3. Assembler Code 4-12
5−1. Watchdog Timer Program 5-7
6−1. ADC Program 6-17
7−1. DAC Loading 7-4
7−2. DAC Program 7-8
8−1. PWM Generator 8-4
9−1. MSC1211 as a Master 9-11
9−2. MSC1211/13 as a Slave 9-13
9−3. MSC1211/13 as an Interrupt-Driven Slave 9-15
10−1. SPI, Simple, Polled Environment 10-9
10−2. SPI FIFO Mode 10-10
11−1. Program Using Timers 0, 1, and 2 11-14
12−1. Serial Port with Software Buffer Code 12-11
13−1. Multiple and Nested Interrupts 13-6
13−2. Wake Up From Idle 13-10

Chapter 1

Introduction

This chapter provides a functional overview of the MSC121x precision analog-to-digital converter (ADC) and digital-to-analog converters (DACs) with 8051 microcontroller and flash memory.

Topic **Page**

1.1	MSC121x Description	1-2
1.2	MSC121x Pinout	1-4
1.3	Enhanced 8051 Core	1-12
1.4	Family Compatibility	1-13
1.5	Flash Memory	1-13
1.6	Internal SRAM	1-13
1.7	High-Performance Analog Functions	1-13
1.8	High-Performance Peripherals	1-14

1.1 MSC121x Description

The MicroSystem family of devices is designed for high-resolution measurement applications in smart transmitters, industrial process control, weigh scales, chromatography, and portable instrumentation. They provide cost-effective, high-performance, mixed-signal solutions. The MicroSystem family not only includes high-performance analog features and digital processing capability, but also integrates many digital peripherals to offer a unique and effective system solution.

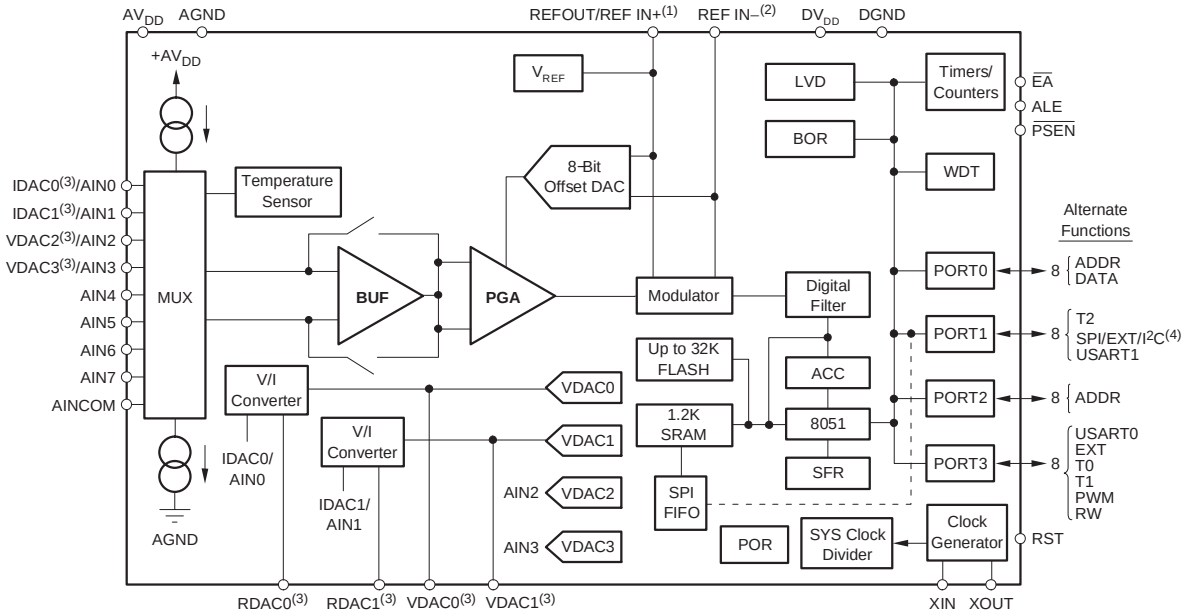
The main components of a MicroSystem product include:

- ❏ High-performance analog functions
- ❏ Low-power enhanced 8051 microcontroller core
- ❏ RAM and Flash memory
- ❏ High-performance digital peripherals

The enhanced 8051 microcontroller includes dual data pointers and executes most instructions up to three times faster than a standard 8051 core. This increased execution speed provides greater flexibility in applications requiring a trade-off among speed, power and noise.

For some designers, the MSC121x is viewed as a microcontroller with integrated analog functions, while to others it is a high-performance analog-to-digital converter (ADC) with an integrated microcontroller. The MSC121x provides unparalleled analog and digital integration for all designers concerned with embedded instrumentation and control.

Figure 1−1. MSC121x Block Diagram



NOTES: (1) On the MSC1210, the REF IN + (pin 30) and REFOUT (pin 31) functions are split onto two pins. On the MSC1211/12/13/14, REFOUT and REF IN+ are combined onto pin 30, and the VDAC1 output is on pin 31.
(2) REF IN− must be tied to AGND when using internal V_{REF}.
(3) DAC functions are only available on the MSC1211/12/13/14.
(4) I^2C is available only on the MSC1211 and MSC1213.

Complementing the high-resolution ADC is a precision voltage reference, programmable gain amplifier (PGA), and analog multiplexer (mux), as well as a temperature sensor and low voltage detectors.

Apart from numerous bit-wise programmable digital ports, there are two USARTs, three timer/counters, a watchdog timer, and a serial (SPI™) bus. Up to 32k of FLASH memory and 1.2K RAM are included as well. The MSC1211/13 also support I^2C serial transfers.

Taken together, the MSC121x features blend analog and digital functions to significantly simplify the overall system design, which reduces the design time and board space, as well as the need for external components.

For systems requiring additional memory, address and data lines are provided via multifunction I/O ports.

This document applies to the following MSC devices:

- ❑ MSC1210
- ❑ MSC1211
- ❑ MSC1212
- ❑ MSC1213
- ❑ MSC1214

For convenience, the abbreviation MSC121x is used to indicate all of the MSC devices listed in this user's guide, unless otherwise specified.

Table 1−1 compares the basic features and functionality of the MSC121x family.

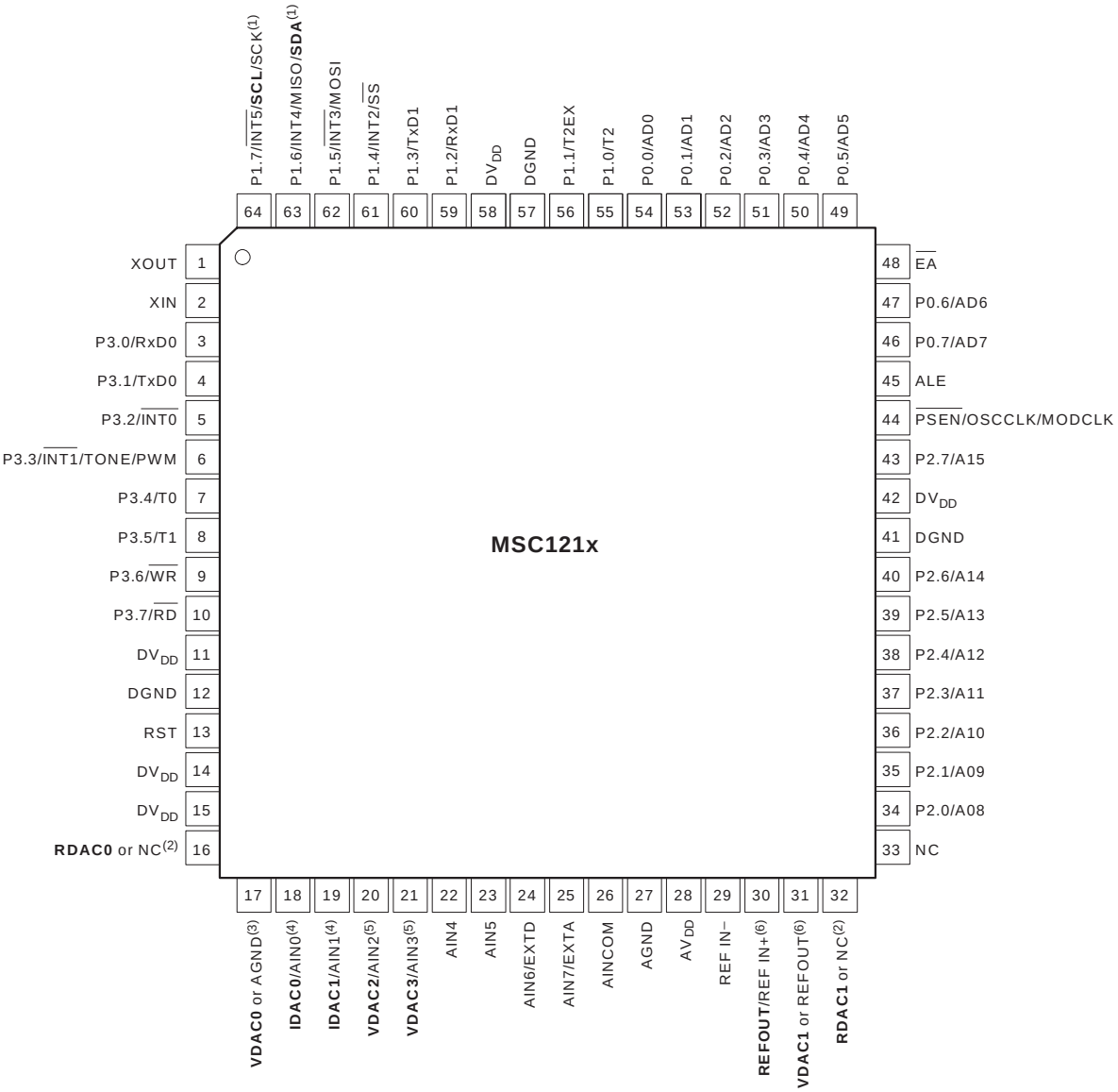
Table 1−1. MSC121x Product Family Matrix

	MSC1210	**MSC1211**	**MSC1212**	**MSC1213**	**MSC1214**
Clock Frequency (kB)	33	33	33	33	33
Flash Memory (kB)	32	32	32	32	32
SRAM (kB)	1.2	1.2	1.2	1.2	1.2
ADC (Channel x Resolution)	8 x 24	8 x 24	8 x 24	8 x 24	8 x 24
DAC (Channel x Resolution)	N/A	Quad Voltage/Dual Current x 16	Quad Voltage/Dual Current x 16	Dual Voltage/Dual Current x 16	Dual Voltage/Dual Current x 16
Features:	34 I/O	34 I/O	34 I/O	34 I/O	34 I/O
32-Bit Accumulator	External Memory	External Memory	External Memory	External Memory	External Memory
Internal V_{REF}	Dual USARTs	Dual USARTs	Dual USARTs	Dual USARTs	Dual USARTs
Internal PGA		I^2C		I^2C	
Internal Buffer	Serial/Parallel	Serial/Parallel	Serial/Parallel	Serial/Parallel	Serial/Parallel
SPI	Programming	Programming	Programming	Programming	Programming
Brownout Reset		System Clock	System Clock	System Clock	System Clock
Low-Voltage Detect		Divider	Divider	Divider	Divider
Package	TQFP-64	TQFP-64	TQFP-64	TQFP-64	TQFP-64

1.2 MSC121x Pinout

The names and functions of pins are similar to those found on most 8051 compatible devices, but with extensions that are specific to the MSC121x. The pin configuration is shown in Figure 1−2, and the pin descriptions are listed in Table 1−2.

Figure 1−2. MSC121x Pin Configuration



NOTES: Non−bolded pin names are on MSC1210.
(1) SCL and SDA not present on MSC1210/12/14.
(2) Pins 16 and 32 are not connected (NC) on MSC1210.
(3) AGND for MSC1210; VDAC0 for MSC1211/12/13/14.
(4) IDAC0 and IDAC1 on MSC1211/12/13/14.
(5) VDAC2 and VDAC3 on MSC1211/12.
(6) For MSC1210, REFOUT is on pin31. For MSC1211/12/13/14, REFOUT is shared with REF IN+ on pin 30, and VDAC1 is on pin 31.

Table 1−2. MSC121x Pin Descriptions

Pin #	Name	Description		
1	XOUT	The output of an oscillator that supports parallel resonant AT cut crystals and ceramic resonators.		
2	XIN	The input to the crystal oscillator that can also be used as an an external clock input.		
3−10	P3.0–P3.7	Port 3 is an 8-bit bidirectional Input/Output port with alternate functions.		
		PORT 3.x	**Alternate Name(s)**	**Alternate Use**
		P3.0	RxD0	Serial port 0 input
		P3.1	TxD0	Serial port 0 output
		P3.2	$\overline{\text{INT0}}$	External interrupt 0
		P3.3	$\overline{\text{INT1}}$/TONE/PWM	External interrupt 1/TONE/PWM output
		P3.4	T0	Timer 0 input
		P3.5	T1	Timer 1 input
		P3.6	$\overline{\text{WR}}$	External data memory write strobe
		P3.7	$\overline{\text{RD}}$	External data memory read strobe
11, 14, 15, 42, 58	DV_{DD}	Digital power supplies. All must be used.		
12, 41, 57	DGND	Digital grounds. All must be used.		
13	RST	A HIGH on the reset input for two clock cycles resets the device.		
		Base MSC1210 Pin Function		**Alternate or Additional in MSC1211/12/13/14**
16	NC	No connection		RDAC0 (MSC1211/12/13/14 only)
17	AGND	Analog ground		VDAC0 (MSC1211/12/13/14 only)
18	AIN0	Analog input channel 0 = AIN0		AIN0 and IDAC0 (MSC1211/12/13/14 only)
19	AIN1	Analog input channel 1 = AIN1		AIN1 and IDAC1 (MSC1211/12/13/14 only)
20	AIN2	Analog input channel 2 = AIN2		AIN2 and VDAC2 (MSC1211/12 only)
21	AIN3	Analog input channel 3 = AIN3		AIN3 and VDAC3 (MSC1211/12 only)
22	AIN4	Analog input channel 4 = AIN4		same
23	AIN5	Analog input channel 5 = AIN5		same
24	AIN6, EXTD	Analog input channel 6 = AIN6 and digital low voltage detect input		same
25	AIN7, EXTA	Analog input channel 7 = AIN7 and analog low voltage detect input		same
26	AINCOM	Analog common for single-ended inputs		same
27	AGND	Analog ground		same
28	AV_{DD}	Analog power supply		same
29	REF IN–	Voltage reference negative input		same
30	REF IN+	Voltage reference positive input		REFOUT/REF IN+
31	REFOUT	Voltage reference output		VDAC1 (MSC1211/12/13/14 only)
32	NC	No connection		RDAC1 (MSC1211/12/13/14 only)
33	NC	No connection		same

Table 1−2. MSC121x Pin Descriptions (continued)

Pin #	Name	Description		
34−40, 43	P2.0−P2.7	Port 2 is an 8-bit bidirectional input/output port with alternate functions.		
		PORT 2.x	**Alternate Name**	**Alternate Use**
		P2.0	A8	Address bit 8
		P2.1	A9	Address bit 9
		P2.2	A10	Address bit 10
		P2.3	A11	Address bit 11
		P2.4	A12	Address bit 12
		P2.5	A13	Address bit 13
		P2.6	A14	Address bit 14
		P2.7	A15	Address bit 15
44	$\overline{\text{PSEN}}$, OSCCLK, MODCLK, Low or High	Program store enable. Connected to optional external memory as a chip enable. $\overline{\text{PSEN}}$ provides an active low pulse. It is used in conjunction with RST and ALE to define serial or parallel programming mode. When not using external program memory, this pin can also be selected to output the oscillator clock, ADC modulator clock, low or high. (See SFR PASEL, F2h.)		
		ALE	**$\overline{\text{PSEN}}$**	**Program Mode Selection (at reset)**
		NC	NC	Normal operation
		0	1	Parallel programming of FLASH
		1	0	Serial programming of FLASH
		0	0	Reserved
45	ALE, Low, High	Address latch enable. Used for latching the low byte of the address during an access to external memory. (See $\overline{\text{PSEN}}$ and SFR PASEL, F2h.)		
48	$\overline{\text{EA}}$	If $\overline{\text{EA}}$ is low as RST falls, *and* neither ALE nor $\overline{\text{PSEN}}$ is low (see above), code access will always be to external memory starting at address 0000h. Otherwise, internal program memory will be accessed where available.		
46, 47, 49−54	P0.0−P0.7	Port 0 is an 8-bit bidirectional input/output port with alternate functions.		
		PORT 0.x	**Alternate Name**	**Alternate Use**
		P0.0	AD0	Address/Data bit 0
		P0.1	AD1	Address/Data bit 1
		P0.2	AD2	Address/Data bit 2
		P0.3	AD3	Address/Data bit 3
		P0.4	AD4	Address/Data bit 4
		P0.5	AD5	Address/Data bit 5
		P0.6	AD6	Address/Data bit 6
		P0.7	AD7	Address/Data bit 7
55, 56, 59−64	P1.0−P1.7	Port 1 is an 8-bit bidirectional input/output port with alternate functions.		
		PORT 1.x	**Alternate Name**	**Alternate Use**
		P1.0	T2	Address/Data bit 0
		P1.1	T2EX	Address/Data bit 1
		P1.2	RxD1	Address/Data bit 2
		P1.3	TxD1	Address/Data bit 3
		P1.4	INT2/$\overline{\text{SS}}$	Address/Data bit 4
		P1.5	$\overline{\text{INT3}}$/MOSI	Address/Data bit 5
		P1.6	INT4/MISO/SDA(1)	Address/Data bit 6
		P1.7	$\overline{\text{INT5}}$/SCL(1)/SCK	Address/Data bit 7

1) SCL and SDA not present on MSC1210/12/14.

1.2.1 Input/Output (I/O) Ports − P0, P1, P2, and P3

In principle, each port consists of eight bits, each of which may be placed low, high, or read by accessing the corresponding bit in the appropriate special function register (SFR). However, when alternate functions are used, the port SFRs are not usually accessed.

Every I/O port bit has an optional pull-up resistor that is enabled when the bit is in 8051-compatible mode (default after RESET), as configured by the P*x*DDRH and P*x*DDRL SFRs, where *x* = 0 to 3. The pull-up resistor is disabled when the port bit is configured as either a CMOS output, open drain output or input, or when accessing external memory, as shown in Figure 1−3 through Figure 1−6.

- When a port pin is to act as an input to an alternate function, it is essential that the pin is not configured as an output.
- To make use of the alternate functions associated with Ports 1 and 3, the corresponding port output latches should be high with the data direction bits defined in a manner appropriate to the alternate function.
- A special case exists for the 8051 mode, which has a weak pull-up resistor and offers bidirectional capability.

Figure 1−3. Standard 8051 I/O Pin Structure

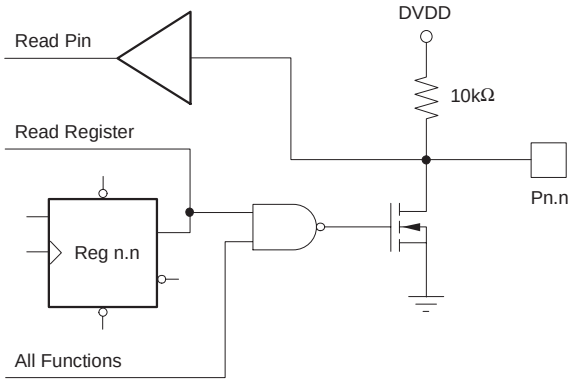


Figure 1−4. CMOS Output Pin Structure

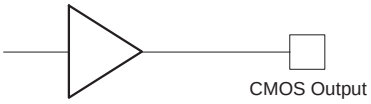


Figure 1−5. Open-Drain Output Pin Structure

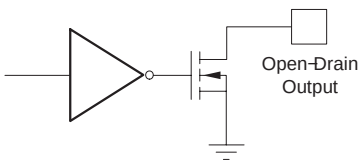
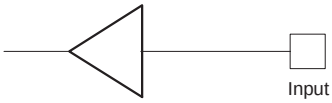


Figure 1−6. Input Pin Structure



1.2.1.1 Port 0 − P0

By default Port 0 provides eight independently-programmable input/output bits. However, it may be configured to provide eight multiplexed low-order address and data lines so that external memory may be accessed. See bit 1 of hardware configuration register 1 (HCR1).

External memory cycles may occur if:

1) The $\overline{EA}$ pin was low when the RST pin was released;

2) An instruction is fetched from an address that is not associated with on-chip FLASH; or

3) When EGP0 (of HCR1) = 0 and a MOVC or MOVX instruction executes.

1.2.1.2 Port 1 − P1

Port 1 provides not only eight independently-programmable bits, but also a variety of alternate functions, as shown in Table 1−3.

Table 1−3. Port 1 Alternate Functions

Port 1 Bit Name	Alternate Function
P1.0 (T2)	Clock source for Timer/Counter 2 when C/$\overline{T}$2 (T2CON.1) is 1.
P1.1 (T2EX)	If Timer/Counter 2 is in auto-reload mode and EXEN2 (T2CON.3) is 1, a negative edge (1−0 transition) causes Timer/Counter 2 to be reloaded and EXF2 (T2CON.6) to be set, which in turn may cause an interrupt.
P1.2 (RxD1)	Serial input to USART1. An external receiver is needed to level shift RS232 signals.
P1.3 (TxD1)	Serial output from USART1. An external driver is needed to level shift RS232 signals.
P1.4 (INT2/$\overline{SS}$)	Positive-edge triggered external 2 interrupt or active-low Slave-Select output during SPI operations.
P1.5 ($\overline{INT3}$/MOSI)	Negative-edge triggered external 3 interrupt or the Master-Out/Slave-In during SPI operations.
P1.6 (INT4/MISO/SDA)	Positive-edge triggered external 4 interrupt or Master-In/Slave-Out during SPI operations, serial data during I^2C operation.
P1.7 ($\overline{INT5}$/SCK/SCL)	Negative-edge triggered external 5 interrupt or serial clock output for SPI operations, serial clock during I^2C operation.

1.2.1.3 Port 2 − P2

By default, Port 2 acts as eight general-purpose input/output signals. However, its alternate function is to provide the upper byte of a 16-bit external address as determined by the $\overline{EA}$ pin and bit 0 (EGP23) of HCR1. If $\overline{EA}$ is low when RST is de-asserted, all memory accesses are external and Port 2 continually outputs the high-order byte of 16-bit addresses. It also outputs bits of an address if EGP23 is 0 and a MOVX instruction is executed, regardless of $\overline{EA}$. This is either the upper byte of the data pointer or the value in MPAGE at 92h, according to whether the MOVX instruction references DPTR or @Rx, respectively.

When $\overline{EA}$ causes external memory accesses, the read and write strobes at P3.7 and P3.6, respectively, are enabled automatically. Selective external accesses must enable these strobes by clearing bit 1 (EGP0) or bit 0 (EGP23) of HCR1 to 0.

If EGP23 = 1 and $\overline{EA}$ = 1, Port 2 output pins are always derived from its data latch .

1.2.1.4 Port 3 − P3

Port 3 provides not only eight independently programmable bits, but also alternate functions, as shown in Table 1−4.

Table 1−4. Port 3 Alternate Functions

Port 1 Bit Name	Alternate Function
P3.0 (RxD0)	Serial input to USART0. An external receiver is needed for RS232 signals.
P3.1 (TxD0)	Serial output from USART0. An external driver is needed for RS232 signals.
P3.2 ($\overline{INT0}$)	Active-low or negative-edge triggered interrupt. Gate for Timer/Counter 0.
P3.3 ($\overline{INT1}$/TONE/PWM)	Active-low or negative-edge triggered interrupt. Tone or Pulse-Width-Modulated output. Gate for Timer/Counter 1.
P3.4 (T0)	Clock source for Timer/Counter 0 if TMOD.2 is 1. See description of the Timer/Counters for gated conditions.
P3.5 (T1)	Used as a clock source for Timer/Counter 1 if TMOD.6 is 1. See description of the Timer/Counters for gated conditions.
P3.6 ($\overline{WR}$)	Active-low write strobe for external memory if used.
P3.7 (RD)	Active-low write strobe for external memory if used.

1.2.2 Oscillator XOUT (pin 1) and XIN (pin 2)

In many applications, a quartz crystal or ceramic resonator is connected between XOUT and XIN to provide a reference clock that is between 1MHz and approximately 30MHz. The static design of the MSC121x allows a digital clock to be applied to XIN that is between 0MHz and 30MHz. A commonly-used crystal for exact baud rates is 11.0592MHz.

Note:

The load capacitors for the crystal must be verified to work over the operating conditions of the application. Due to the design of the oscillator circuit, it is generally better to use lower value load capacitors than those recommended by the crystal manufacturer.

1.2.3 Reset Line − RST (pin 13)

RST is the master reset line. When it is brought high for two or more clock cycles, the MSC121x is reset. All SFRs are placed at their default values and the program counter is reset to 0000h. The contents of internal SRAM are not affected by a reset, and instruction execution begins when RST is brought low, when both $\overline{\text{PSEN}}$ and ALE are high. If either $\overline{\text{PSEN}}$ or ALE is low when RST is brought low, the MSC121x enters Flash Programming mode.

The RST pin has a CMOS Schmitt-trigger input that permits the use of a simple RC network to achieve reset when power is first applied. For the MSC1210, the internal pull-down resistor is typically 200kΩ. For the MSC1211/12/13/14, there is no internal pull-down resistor.

1.2.4 Address Latch Enable − ALE (pin 45)

As RST is de-asserted (low), ALE temporarily acts as an input with a 9kΩ internal pull-up resistor and is used in conjunction with $\overline{\text{PSEN}}$ to place the MSC121x in a programming mode. If neither is pulled low, the MSC121x begins normal operation where ALE is always an output that usually controls a strobed latch to demultiplex the address appearing on Port 0.

When no external memory is present, ALE may be used as an independent output that can be placed low or high via the ALE mode bits in PASEL at F2h.

1.2.5 Program Store Enable − $\overline{\text{PSEN}}$ (pin 44)

As RST is de-asserted (low), $\overline{\text{PSEN}}$ temporarily acts as an input with a 9kΩ internal pull-up resistor, and is used in conjunction with ALE to place the MSC121x in a programming mode. If neither is pulled low, the MSC121x begins normal operation where $\overline{\text{PSEN}}$ is always an output that usually acts as an active-low strobe to read from external program memory.

When no external memory is present, $\overline{\text{PSEN}}$ may be used as an independent output that can be placed low, high, or reflect the ADC modulator clock, via $\overline{\text{PSEN}}$ mode bits in PASEL at F2h.

1.2.6 External Access − $\overline{\text{EA}}$ (pin 48)

$\overline{\text{EA}}$ is sampled as the RST pin is de-asserted (low) and determines whether the MSC121x fetches instruction codes from internal or external memory. When $\overline{\text{EA}}$ is high, code is fetched from internal memory; otherwise, code is always fetched from external memory. Changing the level on $\overline{\text{EA}}$ during normal operation has no effect.

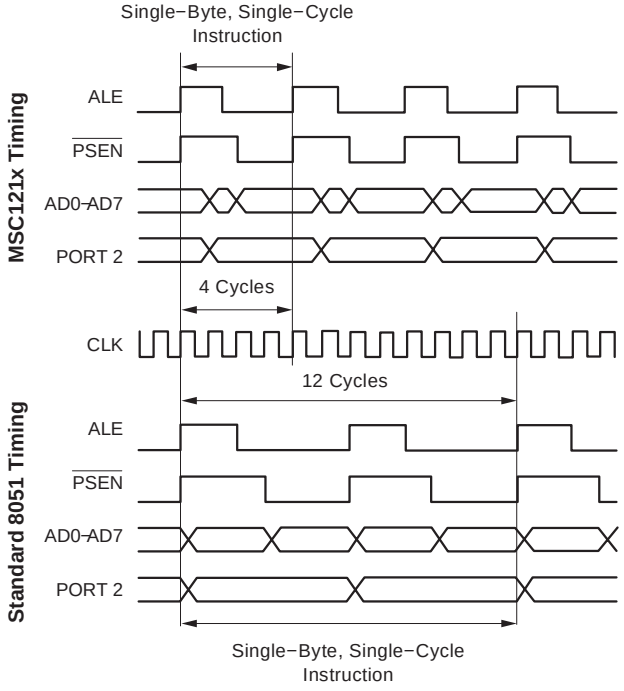
Code is fetched at addresses pointed to by the Program Counter (PC) during program execution and also when a MOVC instruction is executed. In either case, if $\overline{\text{EA}}$ is high but there is no internal memory associated with a particular address, an external fetch will occur.

1.3 Enhanced 8051 Core

All members of the MSC121x family of mixed-signal microcontrollers use a core that is instruction-set-compatible with the industry standard 8051. All instruction codes have the same binary patterns and produce exactly the same logical changes. *However*, the MSC121x is approximately three times faster in execution for the same clock frequency; instead of using 12 clocks per instruction cycle the MSC121x uses four.

The designer can either make use of the increased speed of execution or achieve the same speed, but at a lower clock frequency. A lower clock speed results in less system noise and lower power dissipation.

Figure 1−7. Comparison of MSC121x Timing to Standard 8051 Timing



When porting existing 8051 code to the MSC121x, the designer/programmer may need to consider the change in performance associated with all software timing loops and make adjustments where necessary. By default, hardware timers are still clocked every 12 clock cycles; this may be changed to every four cycles, if required.

Software development tools for the 8051/8052 can be used directly to develop programs for the MSC121x.

1.4 Family Compatibility

The MSC121x family allows the most cost-effective part to be used for each application and ensures a migration path towards larger memories when required. Code written for the 4K byte part runs unaltered on 8K, 16K, and 32K parts. Between the MSC1210 and MSC1211, the allocation and meaning of pins is similar but not identical because of the different functions that are provided. However, among all the MSC121x devices, the parts differ only in the amount of on-chip Flash memory available.

1.5 Flash Memory

The MSC121x parts feature flexible Flash memory that can be partitioned into program and data areas that are best suited for each application. They may be programmed over the entire operating voltage range and temperature range using serial, parallel, and self-programming methodologies.

1.6 Internal SRAM

The MSC121x contains a total of 1280 bytes of static random access memory (RAM). 128 bytes are directly addressable using instructions that incorporate the address. An additional 128 bytes are indirectly addressable via instructions using a register as a pointer, while 1024 bytes are logically external but physically internal and accessed with the MOVX instruction.

1.7 High-Performance Analog Functions

The analog functionality is state-of-the-art. The ADC is extremely low-noise, and meets the most stringent requirements for analog instrumentation. The integrated programmable gain amplifier (PGA) further improves the performance of the ADC, which then achieves nanovolt resolution.

The integrated low-drift high-accuracy voltage reference complements the performance of the ADC and usually eliminates the need for an external reference. However, ratiometric measurements are still possible and easily implemented.

Also present are a programmable filter, an analog multiplexer for single-ended and differential signals, a temperature sensor, burnout current sources, an analog input buffer, and an offset DAC.

1.8 High-Performance Peripherals

Additional digital peripherals are included, which offload CPU processing and control functions from the core to improve further the overall efficiency. In particular, there is a 32-bit accumulator closely associated with the ADC, an SPI-compatible serial port with a FIFO buffer, two USARTs, power-on reset, browout reset, low-voltage detection, multiple digital ports with configurable I/O, a 16-bit pulse-width modulator (PWM), a watchdog timer, and three timer/counters.

The SPI interface and FIFO buffer allow synchronous serial communications with minimal CPU overhead. For the MSC1211 and MSC1213, an I^2C interface may be enabled, which replaces the SPI.

The 32-bit accumulator significantly reduces the processing overhead associated with multi-byte data. It allows automatic 32-bit additions from the ADC, and shifts without using CPU registers. 32-bit addition is supported with minimal program interaction.

Chapter 2

MSC121x Addressable Resources

This chapter provides a detailed description of the MSC121x addressable resources.

Topic **Page**

2.1	Introduction	2-2
2.2	Program Memory and Data Memory	2-3
2.3	Core Data Memory and Special Functions Registers	2-5
2.4	Beyond 64 KBytes	2-6

2.1 Introduction

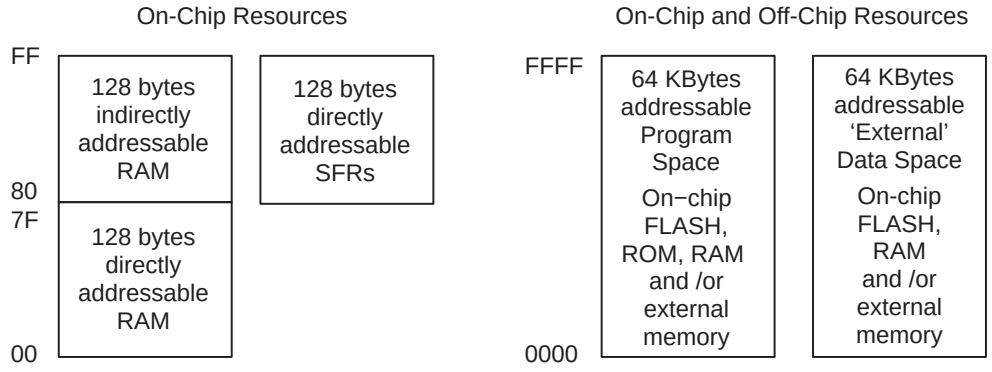
Some microprocessors have a single unified address space that is used for program code, data values and Input/Output ports. However, most 8051 cores (and thus the MSC121x), have several distinct addressable spaces that serve different purposes, as shown in Figure 2−1. In fact, the MSC121x implements all address spaces found in the 8051, but with a feature that permits self-modifiable code.

Direct and indirect 8-bit addresses access up to 384 bytes of on-chip resources, comprised of 256 bytes of static random access memory (SRAM) and up to 128 SFRs. 16-bit pointers (PC and DPTR) allow up to 64 KBytes of program memory and 64 KBytes of extended data memory to be accessed, which may be on-chip and/or off-chip.

Memory for data may be allocated in different places, depending upon the size of the data, how frequently it is altered, and how efficiently it is accessed. The resources available on the MSC121x are:

- 256 bytes of on-chip SRAM for working registers, bit-wide variables, byte and multi-byte variables, and a stack. This memory is accessed by the majority of data-processing instructions.
- 1024 bytes of on-chip extended SRAM, which is considered by the architecture as logically external data. It is used for variables that are needed less frequently and accessed only with MOVX (X for external) instructions, even though it is on-chip.
- A configurable number of KBytes of on-chip FLASH memory that is accessed only with MOVX (X for external) instructions, even though it is on−chip. Typically, data here consists of lookup tables.
- A configurable number of KBytes of user-defined, read-only memory that is off-chip. It is accessed only with MOVX instructions.

Figure 2−1. On-Chip and Off-Chip Resources



Memory for program code may be on-chip or off-chip. On-chip, it is realized by FLASH, ROM, or SRAM within the address range of 0000 to FFFFh. During program execution, if a code address is referenced that is not associated with on-chip memory, off-chip memory will be accessed. Even if on-chip program memory is present, off-chip memory will be used, as long as $\overline{EA}$ is low when the RST (reset) pin is released. $\overline{EA}$ also overrides access to on-chip SRAM that is mapped into code space.

Both program memory and data memory have 16-bit address spaces. They are logically distinct and usually physically separate.

2.2 Program Memory and Data Memory

Table 2−1 shows the addresses associated with program memory and external data memory, which may be located on-chip or off-chip. Accessing off-chip memory requires additional circuitry and the use of numerous pins. Additional memory is gained at the expense of other functions associated with these pins.

Table 2−1. Program Memory and External Data Memory Addresses

<table>
<tr><th colspan="2">Program (Code) Memory(1)</th><th colspan="2">Notes</th><th colspan="2">Data Memory(1)</th></tr>
<tr><td>FFFF</td><td rowspan="2">2K Boot ROM
(if EBR is 1)</td><td rowspan="2">2</td><td rowspan="4">3</td><td>FFFF</td><td rowspan="4">30K off-chip</td></tr>
<tr><td>F800</td><td rowspan="3">8800</td></tr>
<tr><td>F7FF</td><td rowspan="2">28K off-chip</td><td rowspan="2">3</td></tr>
<tr><td>8800</td></tr>
<tr><td>87FF</td><td rowspan="2">1 K on-chip
or off-chip</td><td rowspan="2">4</td><td rowspan="2">4</td><td>87FF</td><td rowspan="2">1K on-chip
or off-chip</td></tr>
<tr><td>8400</td><td>8400</td></tr>
<tr><td>83FF</td><td rowspan="2">1K off-chip</td><td rowspan="2"></td><td rowspan="2">Y5
32K</td><td>83FF</td><td rowspan="2">16K</td></tr>
<tr><td>8000</td><td>4400</td></tr>
<tr><td>7FFF</td><td rowspan="2">16K</td><td rowspan="2">Y5
32K</td><td rowspan="2">Y4
16K</td><td>43FF</td><td rowspan="2">8K</td></tr>
<tr><td>4000</td><td>2400</td></tr>
<tr><td>3FFF</td><td rowspan="2">8K</td><td rowspan="2">Y4
16K</td><td rowspan="2">Y3
8K</td><td>23FF</td><td rowspan="2">4K</td></tr>
<tr><td>2000</td><td>1400</td></tr>
<tr><td>1FFF</td><td rowspan="2">4K</td><td rowspan="2">Y3
8K</td><td rowspan="2">Y2
4K</td><td>13FF</td><td rowspan="2">4 K</td></tr>
<tr><td>1000</td><td>0400</td></tr>
<tr><td>0FFF</td><td rowspan="2">4K</td><td rowspan="2">Y2
4K</td><td rowspan="2">4</td><td>03FF</td><td rowspan="2">1K on-chip
or off-chip</td></tr>
<tr><td>0000</td><td>0000</td></tr>
</table>

1) Y2,Y3,Y4 and Y5 suffix on part code indicates total FLASH of 4K, 8K, 16K and 32K, respectively. This may be partitioned between code and external data spaces, The shaded cells shows the areas that can be defined.

2) All MSC121x devices have 2 KBytes of on-chip Boot ROM. This is enabled by default (see EBR, bit 4 of HCR0) and gives the user program access to a number of useful routines. When disabled, the space is available for external program memory.

3) To permit off-chip expansion, all MSC121x devices have a region for external code and/or data memory; that is, 8800h to F7FFh (boot ROM enabled) or 8800h to FFFFh (boot ROM disabled) for code and 8800h to FFFFh for data.

4) By default, bit 0 of MCON (RAMMAP, SFR 95h) is 0 and 1 KBytes of on-chip SRAM appears only as external data memory between addresses 0000h and 03FFh. If RAMMAP is 1, this SRAM is replicated as code and data at addresses 8400h to 87FFh in user mode.

In typical 8051 architecture, program memory is read-only. However, in the MSC121x, Flash memory that is allocated to code space can be modified when an instruction such as MOVX @DPTR,A is executed with bit 0 of MWS (SFR 8Fh) set to 1. For more details, see the Program Memory Lock and Reset Sector Lock bits in HCR0. Although modifying code in this way can provide flexibility of design, it is not intended to support repetitive use of self-modifying coding techniques. For this purpose, the user may choose to map the 1024 bytes of on-chip SRAM to data and code spaces.

The Boot ROM provides functions to manipulate the Flash memory, but other routines can be copied to code-mapped SRAM.

The on-chip Flash memory may be partitioned so that it is shared between code and data spaces. This is done via the three least-significant bits (DFSEL) in HCR0 when the MSC121x is programmed.

2KB of on-chip Boot ROM is used during serial and parallel programming modes when it is temporarily mapped to 0000h to 07FFh. During normal program execution, it may be mapped into addresses F800h to FFFFh to provide access to useful routines (for example, serial I/O). This occurs by default via bit 4 of HCR0.

Program memory is accessed in an implicit manner as a program is executed, or by explicit use of the assembly-level MOVC instruction.

Data memory is always accessed via the assembly-level MOVX instruction. Even though this mnemonic stands for MOV eXternal, the memory may be on-chip.

Table 2−2. MSC121x Flash Memory Partitioning and Addresses

HCR0 (Binary)	MSC121xY2		MSC121xY3		MSC121xY4		MSC121xY5	
DFSEL	**PM**	**DM**	**PM**	**DM**	**PM**	**DM**	**PM**	**DM**
000	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
001	0 KB	4 KB	0 KB	8 KB	0 KB	16 KB	0 KB	32 KB
	–	0400−13FF	–	0400−23FF	–	0400−43FF	–	0400−83FF
010	0 KB	4 KB	0 KB	8 KB	0 KB	16 KB	16 KB	16 KB
	–	0400−13FF	–	0400−23FF	–	0400−43FF	0000−3FFF	0400−43FF
011	0 KB	4 KB	0 KB	8 KB	8 KB	8 KB	24 KB	8 KB
	–	0400−13FF	–	0400−23FF	0000−1FFF	0400−23FF	0000−5FFF	0400−23FF
100	0 KB	4 KB	4 KB	4 KB	12 KB	4 KB	28 KB	4 KB
	–	0400−13FF	0000−0FFF	0400−13FF	0000−2FFF	0400−13FF	0000−6FFF	0400−13FF
101	2 KB	2 KB	6 KB	2 KB	14 KB	2 KB	30 KB	2 KB
	0000−07FF	0400−0BFF	0000−17FF	0400−0BFF	0000−37FF	0400−0BFF	0000−77FF	0400−0BFF
110	3 KB	1 KB	7 KB	1 KB	15 KB	1 KB	31 KB	1 KB
	0000−0BFF	0400−07FF	0000−1BFF	0400−07FF	0000−3BFF	0400−07FF	0000−7BFF	0400−07FF
111 (default)	4 KB	0 KB	8 KB	0 KB	16 KB	0 KB	32 KB	0 KB
	0000−0FFF	–	0000−1FFF	–	0000−3FFF	–	0000−7FFF	–

1) PM = Program Memory = Code Space; DM = Data Memory = Data Space.

2) Execution from off-chip memory may be forced when pin $\overline{EA}$ is low at reset.

2.3 Program Data Memory and Special Function Registers

The MSC121x has 256 bytes of on-chip SRAM that are closely associated with the core processor, as well as over 100 SFRs, as shown in Table 2−3.

As instructions are executed, the address of SRAM or SFRs is either explicit or implicit, as shown by the examples in Example 2−1.

Table 2−3. On-Chip 8051 Memory

SFR Base (Hex) *Bit-Addressable Bit #*(1)	**C0** *C0−C7*	**C8** *C8−CF*	**D0** *D0−D7*	**D8** *D8−DF*	**E0** *E0−E7*	**E8** *E8−EF*	**F0** *F0−F7*	**F8** *F8−FF*	
SFR Base (Hex) *Bit-Addressable Bit #*(1)	**80** *80−87*	**88** *88−8F*	**90** *90−97*	**98** *98−9F*	**A0** *A0−A7*	**A8** *A8−AF*	**B0** *B0−B7*	**B8** *B8−BF*	
Designation	**Start Address (Hex)**	**Content (Hex)**							**End Address (Hex)**
SFRs	80	128 byte space for SFRs; only directly addressable							FF
SRAM	80	128 bytes of SRAM; only indirectly addressable							FF
SRAM	30	80 bytes of SRAM; directly and indirectly addressable							7F
Bit #(1)	28	40−47	48−4F	50−57	58−5F	60−67	68−6F	70−77	78−7F	2F
Bit #(1)	20	00−07	08−0F	10−17	18−1F	20−27	28−2F	30−37	38−3F	27
Register Bank 3(2)	18	R0	R1	R2	R3	R4	R5	R6	R7	1F
Register Bank 2(2)	10	R0	R1	R2	R3	R4	R5	R6	R7	10
Register Bank 1(2)	08	R0	R1	R2	R3	R4	R5	R6	R7	0F
Register Bank 0(2)	00	R0	R1	R2	R3	R4	R5	R6	R7	07

1) Bit variables numbered 00h to 7Fh are mapped to SRAM bytes 20h to 2Fh. Bit variables numbered 80h to FFh are mapped to SFRs with an address of the form 1xxxx000b. This means that bits within 16 of the 128 possible SFRs may be manipulated by the bit-addressing instructions.

2) Only R0 or R1 may be used as 8-bit indirect pointers to on-chip SRAM between 00h and FFh.

Example 2−1. Instructions

Instruction	Condition or Comment	Net Effect on SRAM or SFR Location
MOV R1,4AH	Register bank 1 is active	Contents of RAM at 4Ah is copied to RAM at 09h
MOV @R1,#F4H	Register bank 2 is active and RAM at address 11h contains 8Ah	Immediate code data of F4h is copied to RAM at 8Ah
SETB sync	sync = 5Eh	Bit 6 of RAM at 2Bh is set
PUSH 34H	Stack Pointer (SP) is 9Bh, but is pre-incremented to 9Ch	Contents of RAM at 34h is copied to RAM at 9Ch
POP P0	P0 = 80h, which is the SFR for physical Port 0; Stack Pointer (SP) is 80h and is post decremented to 7Fh	Contents of RAM at 80h is copied to the SFR at 80h
INC P1	P1 = 90h, which is the SFR for physical Port 1	SFR at 90h is incremented
DEC R6	Register bank 0 is active	Contents of RAM at 06h is decremented
CLC C	C = carry = bit 7 of the Program Status Word at D0h	Bit 7 of SFR at D0h is cleared
MUL AB	Accumulator = 12h, Register B = 3Bh	The accumulator = SFR at E0h, becomes the low part of the product (= 26h), and reg B = SFR at F0h becomes the high part (= 04h)
CPL TF1	TF1 = 8Fh; the bit address of timer 1 overflow flag	Complement bit 7 of SFR TCON at 88h
CLC A		The accumulator at SFR E0h is cleared

1) The Stack Pointer (SP), itself an SFR at 81h, has a default value of 07h. Hence register bank 1, or above, must not be used unless SP is given a higher value.

2) Direct addresses between 80h and FFh always address the SFR space, even when no SFR is defined at a particular location. SRAM between 80h and FFh can only be accessed indirectly or implicitly.

3) The Serial Peripheral Interface (SPI) may be configured to use a circular memory buffer within SRAM and this must be placed so that it does not conflict with other operations.

2.4 Beyond 64 KBytes

If more than 64 KBytes of either program or data storage are required, various bank switching techniques can be used. These may be supported automatically with some C compilers and development environments; refer to the software vendor for further information.

Chapter 3

Special Functions Registers

This chapter describes the special function registers of the MSC121x.

Topic **Page**

3.1	Introduction	3-2
3.2	Referencing SFRs in Assembly and C Languages	3-3
3.3	SFR Types	3-4
3.4	SFR Overview	3-4

3.1 Introduction

Special Function Registers (SFRs) are addressable resources within the MSC121x architecture. They can be accessed by a program in several ways:

1) Via instructions with an 8-bit direct address between 80h and FFh. For example, CLR 80H clears all bits in port 0 to 0.

2) Bit-addressing instructions with bits in the range 80h and FFh. For example, SETB 0A9H enables interrupts from Timer 0.

3) By instructions with implicit access. For example, PUSH 13H increments the Stack Pointer at SFR address 81h before using it as a pointer to save the contents of Register 3 in bank 2.

The 8-bit addresses of all SFRs are shown in Table 3−1 with respect to a base group at addresses of the form 1xxxx000b. The SFRs in this group are byte- and bit-addressable, and shaded in the table.

Reading an unassigned SFR will give 00h, while any values written will be ignored. All SFRs are read and written by the processor one byte at a time, even when they are part of a multi-byte value.

Table 3−1. Special Function Register Map(1)(2)

Base	0 (8)	1 (9)	2 (A)	3 (B)	4 (C)	5 (D)	6 (E)	7 (F)
(F8)	EIP	SECINT	MSINT	USEC	MSECL	MSECH	HMSEC	WDTCON
F0	B	PDCON	PASEL				ACLK	SRST
(E8)	EIE	HWPC0	HWPC1	HDWVER	Reserved	Reserved	FMCON	FTCON
E0	ACC	SSCON	SUMR0	SUMR1	SUMR2	SUMR3	ODAC	LVDCON
(D8)	EICON	ADRESL	ADRESM	ADRESH	ADCON0	ADCON1	ADCON2	ADCON3
D0	PSW	OCL	OCM	OCH	GCL	GCM	GCH	ADMUX
(C8)	T2CON		RCAP2L	RCAP2H	TL2	TH2		
C0	SCON1	SBUF1					EWU	SYSCLK(3)
(B8)	IP							
B0	P3	P2DDRL	P2DDRH	P3DDRL	P3DDRH	DACL(3)	DACH(3)	DACSEL(3)
(A8)	IE	BPCON	BPL	BPH	P0DDRL	P0DDRH	P1DDRL	P1DDRH
A0	P2	PWMCON	PWMLOW TONELOW	PWMHI TONEHI	AIPOL(3)	PAI	AIE	AISTAT
(98)	SCON0	SBUF0	SPICON I2CCON(3)	SPIDATA I2CDATA(3)	SPIRCON I2CGM(3)	SPITCON I2CSTAT(3)	SPISTART I2CSTART(3)	SPIEND
90	P1	EXIF	MPAGE	CADDR	CDATA	MCON		
(88)	TCON	TMOD	TL0	TL1	TH0	TH1	CKCON	MWS
80	P0	SP	DPL0	DPH0	DPL1	DPH1	DPS	PCON

1) In general, the low part of multi-byte SFRs, such as the 16-bit pointer comprised of DPL0 and DPH0, reside at adjacent addresses, but this is not always the case. See TL0 (at 8Ah) and TH0 (at 8Ch).
2) The least significant part of a 16-bit variable is usually at an even address, but this is not always the case. See P2DDRL (at B1h) and P2DDRH (at B2h); P3DDRL (at B3h) and P2DDRH (at B4h); DACL (at B5h) and DACH (at B6h).
3) Refer to the individual product data sheets for information regarding implementation of this function.

3.2 Referencing SFRs in Assembly and C Languages

When writing programs in assembly language, an SFR can be referenced by its absolute address or by a symbol associated with its address. In C language, a variable must first be declared, as shown in Example 3−1.

For assembly language programs, declarations that associate common symbols with values are usually grouped in an *include file* with the name **.inc* that is referenced in the source code. Similarly, for C language, declarations appear in a file with the name **.h*.

Example 3−1.Assembly Code and C Code

Purpose	Assembly Code			C Code (Compiler Dependant Directives)
Output the character ‘A’ to serial port 0	SBUF0	DATA	99H	at 0x99 sfr SBUF0;
	MOV	99H,#41H		–
	MOV	SBUF0,#41H		SBUF0=0x41;
Enable interrupt for Timer 1	IE	DATA	0A8H	at 0xA8 sfr IE;
	ET1	BIT	0ABH	at 0xAB sbit ET1;
	or			or
	ET1	BIT	IE.3	sbit ET1=IE^3;
	SETB	ET1		ET1=1;
Set the decimation ratio for the ADC to 3E8h	decimation	DATA	0DEH	at 0xDE sfr16 decimation
	MOV	decimation,#0E8H		decimation=1000;
	MOV	decimation+1,#3		

1) Indicating a hexadecimal number in assembly language requires a trailing H and leading 0 if the first character would otherwise be a letter; for example, 99H or 099H but 0A8H instead of A8H.

2) In C, a hexadecimal number always starts with 0x; for example, 0x99 and 0xA8.

3) The keyword ‘sfr16’ cannot be used with TH0:TL0 as Timer0 because the addresses are not adjacent. This is also true for TH1:TL1. However, ‘sfr16’ is allowed with TH2:TL2 as Timer2 because TH2 and TL2 are adjacent.

3.3 SFR Types

The SFRs belong to functional groups that relate to different aspects of the operation of the MSC121x.

- ❏ Port Input/Output with bit manipulation
- ❏ Interrupts
- ❏ Integrated peripherals; for example ADC, SPI, USARTS or Counter/Timers
- ❏ System functions; for example, power-down, clock generators and breakpoint registers
- ❏ The core processor architecture; for example, Stack Pointer, Accumulator and Program Status Word
- ❏ Extensions to the architecture; for example, auxiliary data pointer

3.4 SFR Overview

Table 3−2 lists the SFRs, with addresses and descriptions. Underlined SFRs may not be available in all MSC121x devices. Shaded SFR addresses in the table are bit-addressable.

Table 3−2. SFR Overview

Name	Address (Hex)	Description
P0	80h	Port 0 Controls the byte-wide, bit-programmable input/output called Port 0. Each bit in the SFR corresponds to a pin on the actual part. Individual bits may be configured as bidirectional, CMOS output, open drain output, or input via the Data Direction SFRs for Port 0. See P0DDRL at ACh, and P0DDRH at ADh. The same device pins may also be used to provide a multiplexed address and data bus for access to off-chip memory. In this case, bit 1 (EGP0) of HCR1 must be 0 and the program does not reference P0.
SP	81h	Stack Pointer SP acts as an 8-bit pointer to core RAM. It creates a last-in/first-out data structure that is used by the instructions PUSH, POP, ACALL, LCALL, RET, RETI, and interrupt calls. The stack is placed in low memory and grows upwards. SP is pre-incremented and post-decremented, and therefore points to the most recent entry on the stack. The default value is 07h, but this is often increased so that additional register banks may be accessed.
DPL0 DPH0	82h 83h	Data Pointer 0 Low (least significant byte) Data Pointer 0 High (most significant byte) DPL0 and DPH0 are read and written independently (except for the instruction MOV DPTR,#data16), but are used together by instructions that reference the 16-bit data pointer called DPTR. DPTR is used to address code and external data by the MOVC and MOVX instructions, respectively. See Data Pointer Select (DPS) at 86h.

Table 3−2. SFR Overview (continued)

Name	Address (Hex)	Description
DPL1 DPH1	84h 85h	Data Pointer 1 Low (least significant byte) Data Pointer 1 High (most significant byte) DPL1 and DPH1 are read and written independently (except for the instruction MOV DPTR, #data16) but are used together by instructions that reference the 16-bit data pointer called DPTR. DPTR is used to address code and 'external' data by the MOVC and MOVX instructions, respectively. Data Pointer Select (DPS) at 86h.
DPS	86h	Data Pointer Select The original 8051 architecture has one DPTR but the MSC121x has two. If bit 0 of DPS is low, DPTR is formed from DPH0:DPL0; otherwise, it is formed by DPH1:DPL1.
PCON	87h	Power Control The core processor may be placed in low power modes by setting the STOP and IDLE bits of this SFR. It also contains two general-purpose flags, which are often used to help coordinate power-up and power-down activities. There is also a bit called SMOD, which may be used to double the baud rate for serial port 0. This bit is not to be confused with PDCON at F1h, which is used to turn various subsystems on and off.
TCON	88h	Timer Control Bits within TCON control the response to interrupts from Timer/Counters 0 and 1, and external inputs $\overline{INT0}$ and $\overline{INT1}$. Timer/Counters 0 and 1 may also be halted or allowed to run.
TMOD	89h	Timer Mode Configures the modes of operation for Timer/Counters 0 and 1 (for example, whether clocks are internal or external, the number of bits and the reload options). All 8051 Timer/Counters, except for system timers, increment (count up) when they are clocked.
TL0 TH0	8Ah 8Ch	Timer 0 Low Timer 0 High Depending on the mode of operation defined by TMOD at 89h, these SFRs may be considered as independent 8-bit entities, or together as a 13- or 16-bit register. **NOTE:** These SFRs do not have adjacent addresses and cannot be referenced using the 'C' compiler keyword, "sfr16."
TL1 TH1	8Bh 8Dh	Timer 1 Low Timer 1 High Depending on the mode of operation defined by TMOD at 89h, these SFRs may be considered as independent 8-bit entities, or together as a 13-bit, or 16-bit, register. **NOTE:** These SFRs do not have adjacent addresses and cannot be referenced using the C compiler keyword, "sfr16."
CKCON	8Eh	Clock Control The original 8051 required 12 system clock pulses per instruction cycle, and each timer had a divide-by-12 prescaler. Since the MSC121x uses only four clocks, three bits within CKCON selectively allow the prescalers of Timers 0, 1, or 2 to be divide-by-12 (default) or divide-by-4. Three other bits determine the number of wait states introduced into the timing of read ($\overline{RD}$ = P3.7) and write ($\overline{WR}$ = P3.6) strobes when the MOVX instruction is used to access off-chip memory.

Table 3−2. SFR Overview (continued)

Name	Address (Hex)	Description
MWS	8Fh	Memory Write Select When bit 0 is clear (default), any writes to Flash memory via MOVX instructions are written to data space; otherwise, writes are directed to code space. Writing to Flash memory may be inhibited via RSL (bit 5) and PML (bit 6) in HCR0.
P1	90h	Port 1 Controls the byte-wide, bit-programmable input/output called Port 1. Each bit in the SFR corresponds to a pin on the actual part. Individual bits may be configured as bidirectional, CMOS output, open drain output, or input via the Data Direction SFRs for Port 1. See P1DDRL at AEh, and P1DDRH at AFh. Each pin may also be used to provide an alternate function and this may require particular values in P1 and the corresponding data direction bits. For example, P1.4 may be either a general-purpose I/O bit, an input for interrupt INT2 or an active-low Slave Select (input or output) for the SPI interface.
EXIF	91h	External Interrupt Flag Four bits represent interrupt flags for interrupts INT2, $\overline{\text{INT3}}$, INT4 and $\overline{\text{INT5}}$ that must be cleared manually by software. INT2 and INT4 are triggered by a rising edge, while $\overline{\text{INT3}}$ and $\overline{\text{INT5}}$ respond to a negative edge. If a bit is set in software, an interrupt will occur if it is enabled. See Extended Interrupt Enable (EIE) at E8h, and Extended Interrupt Priority (EIP) at FBh
MPAGE	92h	Memory Page During execution of MOVX @R*i*,A or MOVX A,@R*i* an 8-bit, low-order address may be presented to external off-chip data memory via pins associated with Port 0. In the MSC121x, the upper byte of a 16-bit address is placed in MPAGE. This value appears automatically on pins associated with Port 2 when the MOVX instruction is executed.
CADDR	93h	Configuration Address Register The MSC121x contains 128 bytes of Flash memory that may represent hardware configuration data, such as the date of manufacture or any other identification data. This memory is distinct from all other memory addressed by the MSC121x during normal execution of instructions. To access this configuration data, a 7-bit address must first be written to CADDR. See CDATA at 94h.
CDATA	94h	Configuration Data Register Data in the 128 bytes of Flash hardware configuration memory is accessed via this read-only register. The 7-bit address must first be written to CADDR at 093h. **NOTE:** The instruction reading CDATA must not be in Flash memory itself; otherwise, the data read will be invalid. Typically, instructions will be executed from the internal boot ROM, SRAM that is mapped to code space, or off-chip program memory when reading CDATA.
MCON	95h	Memory Configuration Bit 7 is used to identify one of two 16-bit breakpoint registers, while bit 0 determines if the external on-chip RAM is mapped to both code and data spaces or just to data space.
SCON0	98h	Serial Control 0 Contains six bits that determine the format of data on serial port 0 as well as two bits for transmit and receive interrupt flags. It is used in conjunction with TCON at 88h, TMOD at 89h and various timer data registers.
SBUF0	99h	Serial Buffer 0 When written, SFUF0 provides data for the transmitter associated with serial port 0. When read, data is provided by the receive register. Serial data is output on pin TxD0 and received on pin RxD0.

Table 3−2. SFR Overview (continued)

Name	Address (Hex)	Description
SPICON I2CCON	9Ah 9Ah	SPI Control If the Serial Peripheral Interface (SPI) is enabled (see bit 0 of PDCON at F1h), SPICON configures SPI communication characteristics such as data rate, clock polarity, and whether the MSC121x is a master or slave. Writing to SPICON resets the counters and pointers used by the SPI interface in FIFO mode. I2C Control If the I2C interface is enabled (see bit 5 of PDCON at F1h), I2CCON configures I2C communication characteristics, such as START, STOP, ACK, clock stretching, and whether the MSC121x is a master or slave. Writing to I2CCON does not reset the I2C interface.
SPIDATA I2CDATA	9Bh 9Bh	SPI Data If the SPI is enabled (see bit 0 of PDCON at F1h), data written to SPIDATA causes it to be transmitted via the SPI interface, while received data is obtained by reading SPIDATA. I2C DATA If the I2C Interface is enabled (see bit 5 of PDCON at F1h), data written to I2CDATA causes it to be transmitted via the I2C interface, while received data is obtained by reading I2CDATA.
SPIRCON I2CGM	9Ch 9Ch	SPI Receive Control If the SPI is enabled (see bit 0 of PDCON at F1h), SPIRCON defines and monitors the behaviour of the first-in/first-out SPI receive buffer. I2C GM Register If the I2C interface is enabled (see bit 5 of PDCON at F1h), I2CGM determines if a slave MSC1211/13 should respond to a General Call address, or if a master shares a bus with other masters.
SPITCON I2CSTAT	9Dh 9Dh	SPI Transmit Control If the SPI is enabled (see bit 0 of PDCON at F1h), SPITCON defines and monitors the behavior of the SPI transmit buffer and other transmitter features. I2C Status If the I2C Interface is enabled (see bit 5 of PDCON at F1h), I2CSTAT determines the master clock frequency and also the status of the I2C hardware.
SPISTART I2CSTART	9Eh	SPI Buffer Start Address If the SPI is configured to use a circular first-in/first-out buffer, SPISTART specifies the start address in the range 80h to FFh (that is, indirect core SRAM). I2C Start If the I2C interface is enabled (see bit 5 PDCONat F1h), writing to I2CSTART will reset the I2C peripheral to its intitial state.
SPIEND	9Fh	SPI Buffer End Address If the SPI is configured to use a circular first-in/first-out buffer, SPIEND specifies the end address in the range 80h to FFh (that is, indirect core SRAM). SPISTART must be less than SPIEND and together define a buffer from SPISTART to SPIEND, inclusive.
P2	A0h	Port 2 Controls the byte-wide, bit-programmable input/output called Port 2. Each bit in the SFR corresponds to a pin on the actual part. Individual bits may be configured as bidirectional, CMOS output, open drain output, or input via the Data Direction SFRs for Port 2. See P2DDRL (at B1h) and P0DDRH (at B2h). The same device pins may also be used to provide the high-order address for access to off-chip memory. In this case, EGP23 (bit 1) of HCR1 must be 0, and the program should not reference P2.

Table 3−2. SFR Overview (continued)

Name	Address (Hex)	Description
PWMCON	A1h	PWM Control Configures the pulse-width-modulated signal generator. The PWM subsystem is enabled by bit 4 of PDCON at F1h.
PWMLOW TONELOW PWMHI TONEHI	A2h A2h A3h A3h	PWM Low (least significant) Tone Low PWM High (most significant) Tone High PWMHI:PWMLOW or TONEHI:TONELOW represents a 16-bit value for a dedicated counter that is used by the PWM subsystem.
AIPOL	A4h	Auxiliary Interrupt Poll Configures the read operation for AIE and AIPOL (AIE register content or interrupt before masking).
PAI	A5h	Pending Auxiliary Interrupt Provides a 4-bit number that corresponds with the hardware priority of the highest pending auxiliary interrupt. All auxiliary interrupts transfer control to location 0033h.
AIE	A6h	Auxiliary Interrupt Enable Bits written determine if a particular auxiliary interrupt is enabled (not masked). In this group are interrupts from the Seconds timer, ADC Summation, ADC, Milliseconds timer. SPI Transmit, SPI Receive/I^2C Status, Analog Low-Voltage Detect, and Digital Low Voltage Detect. Bits read indicate the status of each auxiliary interrupt before masking (refer to AIPOL at A4h). EIA, bit 5, of EICON at D8h is a common enable for all auxiliary interrupts. See AISTAT at A7h.
AISTAT	A7h	Auxiliary Interrupt Status When read, AISTAT indicates the status of each auxiliary interrupt after masking. A ‘1’ indicates that an interrupt is pending, while a ‘0’ indicates there is either no interrupt or that it is masked. See AIE at A6h.
IE	A8h	Interrupt Enable Bits written determine if a particular interrupt is enabled (not masked). In this group are enables for Serial port 1, Timer 2, Serial port 0, Timer 1, external $\overline{INT1}$, Timer 0 and external $\overline{INT0}$ Bit 7 is a Global Enable for this group of interrupts. Bits read indicate the status of each enable bit (that is, returns what was previously written).
BPCON	A9h	Breakpoint Control Three bits specify the breakpoint conditions. There is a status flag, a bit to select either external data memory or program memory, and another bit to enable an interrupt.
BPL BPH	AAh ABh	Breakpoint Address Low (least significant byte) Breakpoint Address High (most significant byte) BLH:BPL represents the address of 16-bit breakpoint. When this 16-bit address is accessed from either data or code space, as determined by BPCON at A9h, an interrupt may occur. These SFRs are used to assist in real-time debugging.
P0DDRL P0DDRH	ACh ADh	Port 0 Data Direction Low (configures bits 3, 2, 1, and 0 in Port 0) Port 0 Data Direction High (configures bits 7, 6, 5, and 4 in Port 0) Adjacent bits in P0DDRL and P0DDRH control the type of bit presented to device pins by Port 0. Standard 8051 that is bidirectional with weak pull-up is 00, CMOS output is 01, Open drain output is 10 and input only is 11. If EGP0 (bit 1) of HCR1 is 0, or pin $\overline{EA}$ is 0 when the RST pin is released, P0 is either a CMOS input or output, and P0DDRL and P0DDRH have no effect.

Table 3−2. SFR Overview (continued)

Name	Address (Hex)	Description
P1DDRL P1DDRH	AEh AFh	Port 1 Data Direction Low (configures bits 3, 2, 1, and 0 in Port 1) Port 1 Data Direction High (configures bits 7, 6, 5, and 4 in Port 1) Adjacent bits in P1DDRL and P1DDRH control the type of bit presented to device pins by Port 1. Standard 8051 that is bidirectional with weak pull-up is 00, CMOS output is 01, Open drain output is 10 and input only is 11.
P3	B0h	Port 3 Controls the byte-wide, bit-programmable input/output called Port 3. Each bit in the SFR corresponds to a pin on the actual part. Individual bits may be configured as bidirectional, CMOS output, open drain output, or input via the Data Direction SFRs for Port 3. See P3DDRL, at B3h, and P1DDRH, at B4h). Each pin can also be used to provide an alternate function and this may require particular values in P3 and the corresponding data direction bits. For example, P3.0 may be either a general-purpose I/O bit, or an input for serial port 0. If EGP23 (bit 0) or EGP0 (bit 1) of HCR1 are 0, or $\overline{EA}$ is 0 when the RST pin is released, P3.6 is an active-low write strobe, and P3.7 an active-low read strobe. These are used in conjunction with ALE and $\overline{PSEN}$ to coordinate access to off-chip memory.
P2DDRL P2DDRH	B1h B2h	Port 2 Data Direction Low (configures bits 3, 2, 1, and 0 in Port 2) Port 2 Data Direction High (configures bits 7, 6, 5, and 4 in Port 2) Adjacent bits in P2DDRL and P2DDRH control the type of bit presented to device pins by Port 2. Standard 8051 that is bidirectional with weak pull-up is 00, CMOS output is 01, Open drain output is 10 and input only is 11. If EGP23 (bit 0) of HCR1 is 0, or pin $\overline{EA}$ is 0 when the RST pin is released, P2 is either a CMOS input or output, and P2DDRL and P2DDRH have no effect.
P3DDRL P3DDRH	B3h B4h	Port 3 Data Direction Low (configures bits 3, 2, 1, and 0 in Port 3) Port 3 Data Direction High (configures bits 7, 6, 5, and 4 in Port 3) Adjacent bits in P3DDRL and P3DDRH control the type of bit presented to device pins by Port 3. Standard 8051 that is bidirectional with weak pull-up is 00, CMOS output is 01, Open drain output is 10 and input only is 11. If EGP23 (bit 0) or EGP0 (bit 1) of HCR1 are 0, or $\overline{EA}$ is 0 when the RST pin is released, P3.6 is an active-low write strobe, and P3.7 an active-low read strobe. They are CMOS outputs and P3DDRH bits 4 to 7 have no effect.
DACL DACH	B5h B6h	Digital-to-Analog Converter Low (least significant byte) Digital-to-Analog Converter High (most significant byte) DACH:DACL represents 16-bit data values for the four DACs present in the MSC1211. These SFRs are redirected to registers associated with each individual DAC using bits within DACSEL at B7h. Apart from four 16-bit data registers, five different control registers are accessed via DACL, DACH in conjunction with DACSEL.
DACSEL	B7h	Digital-to-Analog Converter Select Writes to DACH and DACL are redirected to other data and control registers according to the least significant three bits of DACSEL. This indirection increases the number of instructions needed to set up all the DACs, but has the benefit that fewer SFR addresses are needed overall.
IP	B8h	Interrupt Priority Bits within IP correspond in position with those enables in IE at A8h. Each determines if the corresponding interrupt has a low or high priority, using 0 or 1 respectively.
SCON1	C0h	Serial Control 1 Contains six bits that determine the format of data on serial port 1, as well as two bits for transmit and receive interrupt flags. It is used in conjunction with TCON at 88h, TMOD at 89h and various timer data registers.

Table 3−2. SFR Overview (continued)

Name	Address (Hex)	Description
SBUF1	C1h	Serial Buffer 1 When written, SBUF1 provides data for the transmitter associated with serial port 1. When read, data is provided by the receive register. Serial data is output on pin TxD1 and received on pin RxD1.
EWU	C6h	Enable Wake-up When the processor has been placed in the IDLE condition by writing a 1 to bit 0 of PCON at 87h, it may be returned to normal operation by an interrupt from either the Watchdog timer, INT1 or INT0. Bits 2, 1, and 0 correspond, in order, with these interrupt sources and act as selective enables when set. An auxiliary interrupt can also restore normal operation; this configuration is enabled with EAI, bit 5, of EICON at D8h.
SYSCLK	C7h	System Clock Divider By default, the crystal oscillator is used as the system clock (that is, $f_{CLK} = f_{OSC}$). SYSCLK allows f_{CLK} to be f_{OSC} divided by 1, 2, 4, 8, 16, 32, 1024, 2048, or 4096 and for the change in the divider to be immediate or synchronized with the milliseconds interrupt. The speed of the processor and all other timers that use f_{CLK} will be affected. When f_{CLK} is decreased, the power consumption is reduced.
T2CON	C8h	Timer Control 2 Timer/Counter 2 is not present in the 8051. It was introduced in the 8052, and therefore, the MSC121x. It has more 16-bit modes of operation than either Timer/Counter 0 or 1. In particular, it offers more resolution when acting as a baud rate generator.
RCAP2L RCAP2H	CAh CBh	Timer 2 Capture Low (least significant byte) Timer 2 Capture High (most significant byte) RCAP2H:RCAP2L form a 16-bit value that is either the value of Timer 2 when in capture mode or the reload value when in auto-reload mode. The function of these SFRs depends on the configuration given in T2CON at C8h*.
TL2 TH2	CCh CDh	Timer 2 Low (least significant byte) Timer 2 High (most significant byte) TH2:TL2 represents the 16-bit value of Timer/Counter 2. The clock source and controls for Timer/Counter 2 are determined by T2CON at C8h*.
PSW	D0h	Program Status Word The processor Program Status Word is accessed via PSW. Bits 7 to 0 represent, in order, Carry, Auxiliary Carry, User Flag 0, Register Bank Select 1 and 0, Overflow Flag, User Flag 1, and Parity Flag. PSW is not saved on the stack automatically at the start of an interrupt service routine (ISR) and it is common for each ISR to begin with the instruction PUSH PSW.
OCL OCM OCH	D1h D2h D3h	(ADC) Offset Calibration Low (least significant byte) (ADC) Offset Calibration Middle (ADC) Offset Calibration High (most significant byte) OCH:OCM:OCL represents a 24-bit value that compensates for the offsets within the ADC or system. Usually values are provided by the ADC subsystem when the ADC is instructed to perform a calibration cycle, but for some applications the user may provide other values.

Table 3−2. SFR Overview (continued)

Name	Address (Hex)	Description
GCL GCM GCH	D4h D5h D6h	(ADC) Gain Low (least significant byte) (ADC) Gain Middle (ADC) Gain High (most significant byte) GCH:GCM:GCL represents a 24-bit value that sets the gain of the ADC or system. Usually values are provided by the ADC subsystem when the ADC is instructed to perform a calibration cycle, but for some applications the user may provide other values.
ADMUX	D7h	ADC Multiplexer Register Selects the sources for the positive and negative inputs of the differential delta-sigma ADC. This includes nine pins on the MSC121x and an internal temperature-related source.
EICON	D8h	Enable Interrupt Control EICON contains the enable for the auxiliary interrupts (EAI), the auxiliary interrupt flag (AI), the watchdog timer interrupt flag (WDTI) and a mode bit for serial port 1 (SMOD1), which doubles the baud rate when set.
ADRESL ADRESM ADRESH	D9h DAh DBh	ADC Conversion Results Low (least significant byte) ADC Conversion Results Medium ADC Conversion Results Low High (most significant byte) ADRESH:ADRESM:ADRESL represents the 24-bit, read-only value of the latest ADC conversion. These registers are not updated on an ADC conversion unless ADRESL has been read from the previous result.
ADCON0	DCh	ADC Control 0 Sets the Burnout Detect, Internal/External voltage reference, 1.25V or 2.5V internal reference, V_{REF} clock source, analog buffer, and programmable gain amplifier, for the delta-sigma ADC.
ADCON1	DDh	ADC Control 1 Sets the polarity, filter type and conversion mode, for the delta-sigma ADC. It also indicates if an overflow or underflow of the summation register has occurred.
ADCON2 ADCON3	DEh DFh	ADC Control 2 ADC Control 3 ADCON3: ADCON2 represent an 11-bit value for the Decimation Ratio of the delta-sigma ADC. The ADC conversion rate is (ACLK+1)/64/Decimation Ratio. See ACLK at F6h.
ACC	E0h	Accumulator The Accumulator is the implicit destination of many operations. Instructions that reference the Accumulator implicitly are always shorter and faster than similar instructions that reference it as an SFR. However, as an SFR it may be used to advantage by instructions such as JB ACC.2,label, or PUSH ACC.
SSCON	E1h	Summation and Shift Control The result of an ADC conversion is placed in ADRES (D9h to DBh), but may be automatically added to a 32-bit sum represented by SUMR (E2h to E5h). The operation of the summation register is controlled by SSCON, which includes the number of times ADC conversions are added to the sum, and the number of bits that the sum is shifted to the right. There is also a mode where 32-bit values provided by the CPU may be added to, or subtracted (MSC1211/12/13/14 only) from, the summation register when SUMR0 is written. If 00h is written to SSCON, the 32-bit summation register is cleared.

Table 3−2. SFR Overview (continued)

Name	Address (Hex)	Description
SUMR0 SUMR1 SUMR2 SUMR3	E2h E3h E4h E5h	Summation Register 0 (least significant byte) Summation Register 1 Summation Register 2 Summation Register 3 (most significant byte) SUMR3:SUMR2:SUMR1:SUMR0 represent the 32-bit sum (and optional shift) of a number of ADC conversions. See SSCON at E1h.
ODAC	E6h	(ADC) Offset DAC Register An analog voltage of up to ± half the range of the ADC is set with an 8-bit DAC and used to offset the input voltage to the ADC. The ODAC register cannot be used to to extend the analog inputs beyond their specified input range.
LVDCON	E7h	Low-Voltage Detection Control The voltages present on the analog and digital supply pins may be enabled to generate interrupts if they fall below preset limits. For either analog or digital, the limits are 2.7V, 3.0V, 3.3V, 4.0V, 4.2V, 4.5V, and 4.7V. The analog low-voltage interrupt may be generated if AIN7 falls below 1.2V, while the digital low-voltage interrupt may be generated if AIN6 falls below 1.2V. This provides a means of detecting an impending power-fail condition while the supply pins themselves are still valid. It can also be used to measure other voltages.
EIE	E8h	Extended Interrupt Enable Provides selective enables for the watchdog timer, $\overline{\text{INT5}}$, INT4, $\overline{\text{INT3}}$, and INT2. See WDTI, bit 3, in EICON at D8h. See External Interrupt Flags, EXIF at 91h.
HWPC0	E9h	Hadware Product Code 0 Refer to the respective data sheet for the code that indicates the type of device.
HWPC1	EAh	Hardware Product Code 1 Refer to the respective data sheet for the code that indicates the type of device.
HDWVER	EBh	Hardware version number Refer to the respective data sheet for the code that indicates the type of device.
FMCON	EEh	Flash Memory Control Three bits to provide control of the Flash memory: specifically, page mode erase, frequency control mode and busy.
FTCON	EFh	Flash Memory Timing Control The upper four bits (FER) determine the Flash erase time while the lower four bits (FWR) determine the Flash write time, according to the following equations: Erase time = $(1 + FER) \times (MSECH:MSECL + 1) \times t_{CLK}$ 5 ms (11ms) for commercial (industrial) temperatures Write time = $5 \times (1 + FWR) \times (USEC + 1) \times t_{CLK}$ The write time should be between 30µs and 40µs. See MSECH and MSECL at FDh and FCh, respectively, and USEC at FBh.
B	F0h	B Register The B register is used only by the instructions MUL AB and DIV AB If these instructions are not used, B is available to store a byte-wide variable or eight single-bit variables. Caution is needed when programming in C because run-time libraries may use these instructions and thus corrupt the value in B.

Table 3−2. SFR Overview (continued)

Name	Address (Hex)	Description
PDCON	F1h	Power-Down Control Active high bits within PDCON provide selective power-down of the subsystems: DAC, I2C, PWM, ADC, Watchdog, System Timer, and SPI.
PASEL	F2h	$\overline{\text{PSEN}}$/ALE Select When off-chip memory is not used control signals, $\overline{\text{PSEN}}$ and ALE are not needed. Three bits in PASEL allow the pin associated with $\overline{\text{PSEN}}$ to be either the usual $\overline{\text{PSEN}}$ signal, system clock, ADC modulator clock, low or high. Two other bits allow the pin associated with ALE to be either the usual ALE signal, low or high. **NOTE:** When these two lines are used as output lines, they should be lightly loaded to avoid entering serial or parallel flash programming modes on reset.
ACLK	F6h	Analog Clock The frequency of the delta-sigma ADC modulator is given by: $\frac{f_{CLK}}{(ACLK + 1) \times 64}$ where f_{CLK} is the frequency of the system clock.
SRST	F7h	System Reset Register If bit 0 is set high then low, the MSC121x will be reset. This causes exactly the same behavior as if a system reset had been initiated by the RST pin. ALE, $\overline{\text{PSEN}}$, and $\overline{\text{EA}}$ will be sampled after the power-up delay.
EIP	F8h	Extended Interrupt Priority Five bits determine the priority for interrupts: Watchdog, $\overline{\text{INT5}}$, INT4, $\overline{\text{INT3}}$, and INT2 See Extended Interrupt Flags, EXIF, at 91h and Extended Interrupt Enable, EIE, at E8h.
SECINT	F9h	Seconds Timer Interrupt The seconds interrupt, if enabled, occurs at an interval given by: $(SECINT + 1) \times (HMSEC + 1) \times (MSEC + 1) \times t_{CLK}$ If bit 7 is set when SECINT is written, the SECINT value will be loaded into the counter immediately; otherwise, it will be delayed until the current count expires. When the associated down-counter reaches zero, it is reloaded with the value in SECINT. See Bit 7,ESEC, of AIE at A6h.
MSINT	FAh	Milliseconds Interrupt The milliseconds interrupt, if enabled, occurs with an interval given by: $(MSINT + 1) \times (MSEC + 1) \times t_{CLK}$ If bit 7 is set when MSINT is written, the MSINT value will be loaded into the counter immediately; otherwise, it will be delayed until the current count expires. When the associated down-counter reaches zero, it is reloaded with the value in MSINT. See Bit 4,EMSEC, of AIE at A6h.
USEC	FBh	Microsecond Register The internal ‘microseconds’ clock has a period given by: $(USEC + 1) \times t_{CLK} = (USEC + 1)/f_{CLK}$ When the associated down-counter reaches zero, it is reloaded with the value in USEC. See FTCON at EFh.

Table 3−2. SFR Overview (continued)

Name	Address (Hex)	Description
MSECL MSECH	FCh FDh	Millisecond Low Millisecond High MSECH:MSECL together represent MSEC, which determines the time between millisecond interrupts given by: $(MSECH \times 256 + MSECL + 1) \times t_{CLK}$ When the associated down-counter reaches zero, it is reloaded with the value in MSECH:MSECL.
HMSEC	FEh	Hundred Millisecond Clock The hundred milliseconds counter has a period given by: $(HMSEC + 1) \times (MSECH \times 256 + MSECL + 1) \times t_{CLK}$ When the associated down-counter reaches zero, it is reloaded with the value in HMSEC.
WDTCON	FFh	Watchdog Control Once enabled, the watchdog timer expires after a delay of: $(WDCNT + 1) \times HMSEC$ to $(WDCNT + 2) \times HMSEC$ Writing a 1, then 0 sequence to bit 7, bit 6, or bit 5 enables, disables, or restarts the watchdog timer, respectively. If the associated down-counter reaches zero, a watchdog timeout occurs. By default, this causes a RESET, but EWDR (bit 3) in HCR0 may disable the reset and trigger an interrupt instead. During normal operation, the counter must be repeatedly restarted before it reaches zero.

Chapter 4

Programmer's Model and Instruction Set

This chapter describes the programmer's model and instruction set for the MSC121x.

Topic **Page**

4.1	Introduction	4-2
4.2	Registers	4-2
4.3	Instruction Types and Addressing Modes	4-4
4.4	MSC121x Op-Code Table	4-10
4.5	Example of MSC121x Instructions	4-12

4.1 Introduction

The MSC121x incorporates a microcontroller that has the same instruction set as the industry-standard 8051; however, for a given external clock, it executes up to three times more quickly. The increased rate is because the MSC121x is based on a machine cycle that consists of four clocks rather than the original 12.

Although the most frequently used instructions are three times faster, the aggregate speed improvement for programs written in C language is about 2.3 times faster.

If application programs are written in C, the programmer has little control over the compiler's choice of instructions; however, for efficient hard real-time operations, assembly-level routines can be achieved that approach a 3x rate increase over the 8051.

4.2 Registers

The MSC121x manipulates data via a single 8-bit accumulator (A), together with eight 8-bit registers (R0 to R7). The result of all arithmetic and logical operations is placed in the accumulator, which can then be copied to R*n*, on-chip memory, or off-chip memory, by various MOV instructions.

To the programmer, the MSC121x may be modelled as shown in Table 4−1.

Table 4−1. 8051 Working Registers

	Bit															
Name	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PSW									CY	AC	F0	RS1	RS0	OV	F1	P
B									Register B							
R*n*									Register *n* (*n* = 0 to 7)							
A									Accumulator							
DPL									Data Pointer Low							
DPH									Data Pointer High							
SP									Stack Pointer							
DPTR	Data Pointer High								Data Pointer Low							
PC	Program Counter															

The Data Pointer (DPTR) is composed of two 8-bit SFRs accessed as separate bytes. However, it is used implicitly by some instructions as a 16-bit pointer to either code or data memory.

The Stack Pointer (SP) is used to support a first-in/first-out data structure within core data memory. When referenced implicitly as an 8-bit pointer, it is pre-incremented and post-decremented, which means that the stack grows upwards and SP always points to the most recent entry. The stack can store either data values or addresses.

The default value for SP is 7, so it starts to grow just above memory associated with address bank 0 at core data memory locations 00h to 07h. Since address bank 1 occupies locations 08h to 0Fh, care must be taken to redefine the initial value of SP whenever register bank 1 (or 2 or 3) is to be used. The selection of the active register bank is determined by bits 4 and 3 of the Program Status Word (PSW). For example, when RS1 = 1 and RS0 = 1, bank 3 is active and R2 corresponds with core data memory location 1Ah.

PSW also contains the Carry flag (CY), Auxiliary Carry (AC), General-purpose flags F1 and F0, as well as the Overflow flag (OV) and the Parity flag (P). Some instructions change the flags, but the majority do not.

Register B is sometimes useful to store a byte-wide variable or 8 bit-wide variables, especially for applications written entirely in assembly language. Care is needed because it is used by MUL and DIV instructions that may be called by C run-time libraries.

The 16-bit program counter (PC) is incremented as sequential instructions are executed. For jumps, it is loaded with a new value, and for CALLs and interrupts, it is stored to the stack for recovery during RETurns and RETI. It always points to a byte in program memory and has a reset value of 0000h.

4.3 Instruction Types and Addressing Modes

MSC121x instruction types are shown in Example 4−1.

For each type of instruction, there may be more than one mode of addressing. For example, there are four different modes associated with the ADD instruction, as shown in Example 4−2.

The MOV instruction has the greatest number of combinations of addressing modes, with special variants such as MOVX and MOVC.

Table 4−3 shows all instructions with their mnemonic, description, flags, cycles, clocks, and op-code. If the exact operation is unclear, the reader is referred to any of the numerous data sheets and books for the 8051 that are generally available.

Example 4−1. Instruction Types

Type	Examples		
Simple data movement	MOV A,R5	MOV R4,#0A3H	MOV P1,A
Data movement	MOV A,@R1	MOVX A,@DPTR	PUSH PSW
Data processing	DEC R3	ADDC A,10H	ORL P2,#5
Bit operations	CLR C	SETB 084H	ANL C,/F0
Program Flow	LCALL *16-bit address*	SJMP *relative address*	CJNE A,#4, *address*
Miscellaneous	DJNZ R4, *relative address*	MUL	DA

Example 4−2. Instruction Addressing Modes

Assembly level instruction	Addressing Mode	Action	Hexadecimal Operation Code(s)
ADD A,#0C3H	Immediate	The code byte at PC + 1 (that is, C3h) is added to A.	24 C3h
ADD A,@R1	Indirect	The contents of Register 1 provide the 8-bit address of the data in core memory that is added to A.	27h
ADD A,P0	Direct	The code byte at PC + 1 provides the 8-bit address of the data in core memory that is added to A. In this case, SFR P0 at 80h.	25 80h
ADD A,R4	Register	The contents of Register 4 is added to the accumulator. The core memory location corresponding to register 4 is either 04h, 0Ch, 14h, or 1Ch depending on the register bank select bits in PSW.	2Ch

Table 4−2. Symbol Descriptions for Instruction List (Table 4−3)

Symbol	Description
A	Accumulator.
R*n*	Register R0−R7 of the current register bank.
direct	Internal core address. RAM (00h−7Fh) or SFR (80h−FFh).
@R*i*	R0 or R1 act as an 8-bit pointer to internal core RAM (00h−FFh), except MOVX references external data space.
rel	Two's complement offset byte (−128 to +127) relative to the start address of the next sequential instruction.
bit	Direct bit address. Bits 00h−7Fh map to RAM while 80h−FFh map to SFRs.
#data	8-bit immediate constant.
#data16	16-bit immediate constant.
addr16	16-bit destination address anywhere within program memory address space.
addr11	11-bit destination address anywhere within the current 2K page of program memory.

Table 4−3. Instruction List

Mnemonic	Description	Flags(1)			Bytes	MSC121x Cycles	MSC121x Clocks	8051 Clocks	Code (Hex)
		CY	AC	OV					
Arithmetic									
ADD A,R*n*	Add register to A	X	X	X	1	1	4	12	28–2F
ADD A,direct	Add direct byte to A	X	X	X	2	2	8	12	25
ADD A,@R*i*	Add indirect data memory to A	X	X	X	1	1	4	12	26–27
ADD A,#data	Add immediate data to A	X	X	X	2	2	8	12	24
ADDC A,R*n*	Add register to A with carry	X	X	X	1	1	4	12	38–3F
ADDC A,direct	Add direct byte to A with carry	X	X	X	2	2	8	12	35
ADDC A,@R*i*	Add indirect data memory to A with carry	X	X	X	1	1	4	12	36–37
ADDC A,#data	Add immediate data to A with carry	X	X	X	2	2	8	12	34
SUBB A,R*n*	Subtract register from A with borrow	X	X	X	1	1	4	12	98–9F
SUBB A,direct	Subtract direct byte from A with borrow	X	X	X	2	2	8	12	95
SUBB A,@R*i*	Subtract indirect data memory from A with borrow	X	X	X	1	1	4	12	96–97
SUBB A,#data	Subtract immediate data from A with borrow	X	X	X	2	2	8	12	94
INC A	Increment A	−	−	−	1	1	4	12	04
INC R*n*	Increment register	−	−	−	1	1	4	12	08–0F
INC direct	Increment direct byte	−	−	−	2	2	8	12	05
INC @R*i*	Increment indirect data memory	−	−	−	1	1	4	12	06–07
DEC A	Decrement A	−	−	−	1	1	4	12	14
DEC R*n*	Decrement register	−	−	−	1	1	4	12	18–1F
DEC direct	Decrement direct byte	−	−	−	2	2	8	12	15
DEC @R*i*	Decrement indirect data memory	−	−	−	1	1	4	12	16–17
INC DPTR	Increment 16-bit data pointer	−	−	−	1	3	12	24	A3
MUL AB	Multiply A by B	0	−	X	1	5	20	48	A4

1) Flags CY, AC, and OV may also be changed by explicit writes to corresponding bits in the PSW.
2) Number of cycles is user-selectable. See SFR CKCON at 8Eh.

Table 4−3. Instruction List (continued)

Mnemonic	Description	Flags(1) CY	AC	OV	Bytes	MSC121x Cycles	MSC121x Clocks	8051 Clocks	Code (Hex)
DIV AB	Divide A by B	0	–	X	1	5	20	48	84
DA A	Decimal adjust A to give 2 BCD nibbles. Used after ADD or ADDC.	X	–	–	1	1	4	12	D4
Logical									
ANL A,R*n*	AND register to A	–	–	–	1	1	4	12	58–5F
ANL A,direct	AND direct byte to A	–	–	–	2	2	8	12	55
ANL A,@R*i*	AND indirect data memory to A	–	–	–	1	1	4	12	56–57
ANL A,#data	AND immediate data to A	–	–	–	2	2	8	12	54
ANL direct,A	AND A to direct byte	–	–	–	2	2	8	12	52
ANL direct,#data	AND immediate data to direct byte	–	–	–	3	3	12	24	53
ORL A,R*n*	OR register to A	–	–	–	1	1	4	12	48–4F
ORL A,direct	OR direct byte to A	–	–	–	2	2	8	12	45
ORL A,@R*i*	OR indirect data memory to A	–	–	–	1	1	4	12	46–47
ORL A,#data	OR immediate data to A	–	–	–	2	2	8	12	44
ORL direct,A	OR A to direct byte	–	–	–	2	2	8	12	42
ORL direct,#data	OR immediate data to direct byte	–	–	–	3	3	12	24	43
XRL A,R*n*	Exclusive OR register to A	–	–	–	1	1	4	12	68–6F
XRL A,direct	Exclusive OR direct byte to A	–	–	–	2	2	8	12	65
XRL A,@R*i*	Exclusive OR indirect data memory to A	–	–	–	1	1	4	12	66–67
XRL A,#data	Exclusive OR immediate data to A	–	–	–	2	2	8	12	64
XRL direct,A	Exclusive OR A to direct byte	–	–	–	2	2	8	12	62
XRL direct,#data	Exclusive OR immediate data to direct byte	–	–	–	3	3	12	24	63
CLR A	Clear A	–	–	–	1	1	4	12	E4
CPL A	Complement A	–	–	–	1	1	4	12	F4
RL A	Rotate A left	–	–	–	1	1	4	12	23
RLC A	Rotate A left through carry	X	–	–	1	1	4	12	33
RR A	Rotate A right	–	–	–	1	1	4	12	03
RRC A	Rotate A right through carry	X	–	–	1	1	4	12	13
SWAP A	Swap nibbles of A	–	–	–	1	1	4	12	C4

1) Flags CY, AC, and OV may also be changed by explicit writes to corresponding bits in the PSW.
2) Number of cycles is user-selectable. See SFR CKCON at 8Eh.

Table 4−3. Instruction List (continued)

Mnemonic	Description	Flags(1)			Bytes	MSC121x Cycles	MSC121x Clocks	8051 Clocks	Code (Hex)
		CY	AC	OV					
Data Movement									
MOV A,R*n*	Move register to A	−	−	−	1	1	4	12	E8–EF
MOV A,direct	Move direct byte to A	−	−	−	2	2	8	12	E5
MOV A,@R*i*	Move indirect data memory to A	−	−	−	1	1	4	12	E6–E7
MOV A,#data	Move immediate data to A	−	−	−	2	2	8	12	74
MOV R*n*,A	Move A to register	−	−	−	1	1	4	12	F8–FF
MOV R*n*,direct	Move direct byte to register	−	−	−	2	2	8	24	A8–AF
MOV R*n*,#data	Move immediate data to register	−	−	−	2	2	8	12	78–7F
MOV direct,A	Move A to direct byte	−	−	−	2	2	8	12	F5
MOV direct,R*n*	Move register to direct byte	−	−	−	2	2	8	24	88–8F
MOV direct,direct	Move direct byte to direct byte	−	−	−	3	3	12	24	85
MOV direct,@R*i*	Move indirect data memory to direct byte	−	−	−	2	2	12	24	86–87
MOV direct,#data	Move immediate data to direct byte	−	−	−	3	3	12	24	75
MOV @R*i*,A	MOV A to indirect data memory	−	−	−	1	1	4	12	F6–F7
MOV @R*i*,direct	Move direct byte to indirect data memory	−	−	−	2	2	8	24	A6–A7
MOV @R*i*,#data	Move immediate data to indirect data memory	−	−	−	2	2	8	12	76–77
MOV DPTR,#data	Move 2 bytes of immediate data to data pointer	−	−	−	3	3	12	24	90
MOVC A,@A+DPTR	Move a byte in code space A (unsigned) after DPTR to A	−	−	−	1	3	12	24	93
MOVC A,@A+PC	Move a byte in code space A (unsigned) after from the address of the next instruction to A	−	−	−	1	3	12	24	83
MOVX A,@R*i*	Move external data (A8) to A	−	−	−	1	2–9(2)	8	12	E2–E3
MOVX A,@DPTR	Move external data (A16) to A	−	−	−	1	2–9(2)	8	24	E0
MOVX @R*i*,A	Move A to external data. Upper 8-bit address comes from MPAGE SFR.	−	−	−	1	2–9(2)	8	24	F2–F3
MOVX @DPTR,A	Move A to external data memory	−	−	−	1	2–9(2)	8	24	F0
PUSH direct	Push direct byte onto stack	−	−	−	2	2	8	24	C0
POP direct	Pop direct byte from stack	−	−	−	2	2	8	24	D0
XCH A,R*n*	Exchange A and register	−	−	−	1	1	4	12	C8–CF
XCH A,direct	Exchange A and direct byte	−	−	−	2	2	8	12	C5
XCH A,@R*i*	Exchange A and indirect data memory	−	−	−	1	1	4	12	C6–C7
XCHD A,@R*i*	Exchange A and indirect data memory nibble in bits 3−0	−	−	−	1	1	4	12	D6–D7

1) Flags CY, AC, and OV may also be changed by explicit writes to corresponding bits in the PSW.
2) Number of cycles is user-selectable. See SFR CKCON at 8Eh.

Table 4−3. Instruction List (continued)

Mnemonic	Description	Flags(1) CY	AC	OV	Bytes	MSC121x Cycles	MSC121x Clocks	8051 Clocks	Code (Hex)
Boolean									
CLR C	Clear carry	0	–	–	1	1	4	12	C3
CLR bit	Clear direct bit	–	–	–	2	2	8	12	C2
SETB C	Set carry	1	–	–	1	1	4	12	D3
SETB bit	Set direct bit	–	–	–	2	2	8	12	D2
CPL C	Complement carry	X	–	–	1	1	4	12	B3
CPL bit	Complement direct bit	–	–	–	2	2	8	12	B2
ANL C,bit	AND direct bit to carry	X	–	–	2	2	8	24	82
ANL C,/bit	AND inverse of direct bit to carry	X	–	–	2	2	8	24	B0
ORL C,bit	OR direct bit to carry	X	–	–	2	2	8	24	72
ORL C,/bit	OR inverse of direct bit to carry	X	–	–	2	2	8	24	A0
MOV C,bit	Move direct bit to carry	X	–	–	2	2	8	12	A2
MOV bit,C	Move carry to direct bit	–	–	–	2	2	8	24	92
Branching									
ACALL addr11	Absolute call to subroutine within current page.	–	–	–	2	3	12	24	11–F1
LCALL addr16	Long call to subroutine; PC becomes addr16	–	–	–	3	4	16	24	12
RET	Return from subroutine	–	–	–	1	4	16	24	22
RETI	Return from interrupt	–	–	–	1	4	16	24	32
AJMP addr11	Absolute unconditional jump within current page.	–	–	–	2	3	12	24	01–E1
LJMP addr16	Long jump; PC becomes addr16	–	–	–	3	4	16	24	02
SJMP rel	Unconditional relative jump	–	–	–	2	3	12	24	80
JC rel	Jump relative if carry is 1	–	–	–	2	3	12	24	40
JNC rel	Jump relative if carry is 0	–	–	–	2	3	12	24	50
JB bit,rel	Jump relative if direct bit is 1	–	–	–	3	4	16	24	20
JNB bit,rel	Jump relative if direct bit is 0	–	–	–	3	4	16	24	30
JBC bit,rel	Jump relative if direct bit is 1 and clear the bit	–	–	–	3	4	16	24	10
JMP @A+DPTR	Jump indirect. Program counter becomes DPTR plus A.	–	–	–	1	3	12	24	73
JZ rel	Jump relative if accumulator is 00h.	–	–	–	2	3	12	24	60
JNZ rel	Jump relative if accumulator is not 00h.	–	–	–	2	3	12	24	70
CJNE A,direct,rel	Compare A with direct data and jump relative if not equal.	X	–	–	3	4	16	24	B5
CJNE A,#data,rel	Compare A with immediate data and jump relative if not equal.	X	–	–	3	4	16	24	B4

1) Flags CY, AC, and OV may also be changed by explicit writes to corresponding bits in the PSW.
2) Number of cycles is user-selectable. See SFR CKCON at 8Eh.

Table 4−3. Instruction List (continued)

Mnemonic	Description	Flags(1)			Bytes	MSC121x Cycles	MSC121x Clocks	8051 Clocks	Code (Hex)
		CY	AC	OV					
CJNE R*n*,#data,rel	Compare register with immediate data and jump relative if not equal.	X	−	−	3	4	16	24	B8–BF
CJNE @R*i*,#data,rel	Compare indirect with immediate data and jump relative if not equal.	X	−	−	3	4	16	24	B6–B7
DJNZ R*n*,rel	Decrement register and jump relative if not 0.	−	−	−	2	3	12	24	D8–DF
DJNZ direct,rel	Decrement direct byte and jump relative if not 0.	−	−	−	3	4	16	24	D5
Miscellaneous									
NOP	No operation	−	−	−	1	1	4	12	00
Reserved	No operation	−	−	−	1	1	4	12	A5

1) Flags CY, AC, and OV may also be changed by explicit writes to corresponding bits in the PSW.
2) Number of cycles is user-selectable. See SFR CKCON at 8Eh.

4.4 MSC121x Op-Code Table

Table 4−4. MSC121x Op-Codes

Table Cell Contents:

opcode	bytes/cycles
instruction	
operand(s)	

Operand Definitions:

addr11: 11-bit address
addr16: 16-bit address
bit: addressable bit
dir: direct address
#d8: 8-bit immediate data
#d16: 16-bit immediate data
rel8: 8-bit relative address

00 1/1 NOP	01 2/3 AJMP addr11	02 3/4 LJMP addr16	03 1/1 RR A	04 1/1 INC A	05 2/2 INC dir	06 1/1 INC @R0	07 1/1 INC @R1
10 3/4 JBC bit,rel8	11 2/3 ACALL addr11	12 3/4 LCALL addr16	13 1/1 RRC A	14 1/1 DEC A	15 2/2 DEC dir	16 1/1 DEC @R0	17 1/1 DEC @R1
20 3/4 JB bit,rel8	21 2/3 AJMP addr11	22 1/4 RET	23 1/1 RL A	24 2/2 ADD A,#d8	25 2/2 ADD A,dir	26 1/1 ADD A.@R0	27 1/1 ADD A,@R1
30 3/4 JNB bit,rel8	31 2/3 ACALL addr11	32 1/4 RETI	33 1/1 RLC A	34 2/2 ADDC A,#d8	35 2/2 ADDC A,dir	36 1/1 ADDC A.@R0	37 1/1 ADDC A,@R1
40 2/3 JC rel8	41 2/3 AJMP addr11	42 2/2 ORL dir,A	43 3/3 ORL dir,#d8	44 2/2 ORL A,#d8	45 2/2 ORL A,dir	46 1/1 ORL A.@R0	47 1/1 ORL A,@R1
50 2/3 JNC rel8	51 2/3 ACALL addr11	52 2/2 ANL dir,A	53 3/3 ANL dir,#d8	54 2/2 ANL A,#d8	55 2/2 ANL A,dir	56 1/1 ANL A.@R0	57 1/1 ANL A,@R1
60 2/3 JZ rel8	61 2/3 AJMP addr11	62 2/2 XRL dir,A	63 3/3 XRL dir,#d8	64 2/2 XRL A,#d8	65 2/2 XRL A,dir	66 1/1 XRL A,@R0	67 1/1 XRL A,@R1
70 2/3 JNZ rel8	71 2/3 ACALL addr11	72 2/2 ORL C,bit	73 1/3 JMP @A+DPTR	74 2/2 MOV A,#d8	75 3/3 MOV dir,#d8	76 2/2 MOV @R0,#d8	77 2/2 MOV @R1,#d8
80 2/3 SJMP rel8	81 2/3 AJMP addr11	82 2/2 ANL C,bit	83 1/3 MOVC A,@A+PC	84 1/5 DIV AB	85 3/3 MOV dir,dir	86 2/2 MOV dir,@R0	87 2/2 MOV dir,@R1
90 3/3 MOV DPTR,#d16	91 2/3 ACALL addr11	92 2/2 MOV bit,C	93 1/3 MOVC A,@A+DPTR	94 2/2 SUBB A,#d8	95 2/2 SUBB A,dir	96 1/1 SUBB A,@R0	97 1/1 SUBB A,@R1
A0 2/2 ORL C,/bit	A1 2/3 AJMP addr11	A2 2/2 MOV C,bit	A3 1/3 INC DPTR	A4 1/5 MUL AB	A5 1/1 NOP	A6 2/2 MOV @R0,dir	A7 2/2 MOV @R1,dir
B0 2/2 ANL C,/bit	B1 2/3 ACALL addr11	B2 2/2 CPL bit	B3 1/1 CPL C	B4 3/4 CJNE A,#d8,rel8	B5 3/4 CJNE A,dir,rel8	B6 3/4 CJNE @R0,#d8,rel8	B7 3/4 CJNE @R1,#d8,rel8
C0 2/2 PUSH dir	C1 2/3 AJMP addr11	C2 2/2 CLR bit	C3 1/1 CLR C	C4 1/1 SWAP A	C5 2/2 XCH A,dir	C6 1/1 XCH A,@R0	C7 1/1 XCH A,@R1
D0 2/2 POP dir	D1 2/3 ACALL addr11	D2 2/2 SETB bit	D3 1/1 SETB C	D4 1/1 DA A	D5 3/4 DJNZ dir,rel8	D6 1/1 XCHD A,@R0	D7 1/1 XCHD A,@R1
E0 1/2−9* MOVX A,@DPTR	E1 2/3 AJMP addr11	E2 1/2−9* MOVX A,@R0	E3 1/2−9* MOVX A,@R1	E4 1/1 CLR A	E5 2/2 MOV A,dir	E6 1/1 MOV A,@R0	E7 1/1 MOV A,@R1
F0 1/2−9* MOVX @DPTR,A	F1 2/3 ACALL addr11	F2 1/2−9* MOVX @R0,A	F3 1/2−9* MOVX @R1,A	F4 1/1 CPL A	F5 2/2 MOV dir,A	F6 1/1 MOV @R0,A	F7 1/1 MOV @R1,A

Table 4−4. MSC121x Op-Codes (continued)

Table Cell Contents:

opcode	bytes/cycles
instruction	
operand(s)	

Operand Definitions:

addr11: 11-bit address
addr16: 16-bit address
bit: addressable bit
dir: direct address
#d8: 8-bit immediate data
#d16: 16-bit immediate data
rel8: 8-bit relative address

08 1/1 INC R0	09 1/1 INC R1	0A 1/1 INC R2	0B 1/1 INC R3	0C 1/1 INC R4	0D 1/1 INC R5	0E 1/1 INC R6	0F 1/1 INC R7
18 1/1 DEC R0	19 1/1 DEC R1	1A 1/1 DEC R2	1B 1/1 DEC R3	1C 1/1 DEC R4	1D 1/1 DEC R5	1E 1/1 DEC R6	1F 1/1 DEC R7
28 1/1 ADD A,R0	29 1/1 ADD A,R1	2A 1/1 ADD A,R2	2B 1/1 ADD A,R3	2C 1/1 ADD A,R4	2D 1/1 ADD A,R5	2E 1/1 ADD A,R6	2F 1/1 ADD A,R7
38 1/1 ADDC A,R0	39 1/1 ADDC A,R1	3A 1/1 ADDC A,R2	3B 1/1 ADDC A,R3	3C 1/1 ADDC A,R4	3D 1/1 ADDC A,R5	3E 1/1 ADDC A,R6	3F 1/1 ADDC A,R7
48 1/1 ORL A,R0	49 1/1 ORL A,R1	4A 1/1 ORL A,R2	4B 1/1 ORL A,R3	4C 1/1 ORL A,R4	4D 1/1 ORL A,R5	4E 1/1 ORL A,R6	4F 1/1 ORL A,R7
58 1/1 ANL A,R0	59 1/1 ANL A,R1	5A 1/1 ANL A,R2	5B 1/1 ANL A,R3	5C 1/1 ANL A,R4	5D 1/1 ANL A,R5	5E 1/1 ANL A,R6	5F 1/1 ANL A,R7
68 1/1 XRL A,R0	69 1/1 XRL A,R1	6A 1/1 XRL A,R2	6B 1/1 XRL A,R3	6C 1/1 XRL A,R4	6D 1/1 XRL A,R5	6E 1/1 XRL A,R6	6F 1/1 XRL A,R7
78 2/2 MOV R0,#d8	79 2/2 MOV R1,#d8	7A 2/2 MOV R2,#d8	7B 2/2 MOV R3,#d8	7C 2/2 MOV R4,#d8	7D 2/2 MOV R5,#d8	7E 2/2 MOV R6,#d8	7F 2/2 MOV R7,#d8
88 2/2 MOV dir,R0	89 2/2 MOV dir,R1	8A 2/2 MOV dir,R2	8B 2/2 MOV dir,R3	8C 2/2 MOV dir,R4	8D 2/2 MOV dir,R5	8E 2/2 MOV dir,R6	8F 2/2 MOV dir,R7
98 1/1 SUBB A,R0	99 1/1 SUBB A,R1	9A 1/1 SUBB A,R2	9B 1/1 SUBB A,R3	9C 1/1 SUBB A,R4	9D 1/1 SUBB A,R5	9D 1/1 SUBB A,R6	9F 1/1 SUBB A,R7
A8 2/2 MOV R0,dir	A9 2/2 MOV R1,dir	AA 2/2 MOV R2,dir	AB 2/2 MOV R3,dir	AC 2/2 MOV R4,dir	AD 2/2 MOV R5,dir	AE 2/2 MOV R6,dir	AF 2/2 MOV R7,dir
B8 3/4 CJNE R0,#d8,rel8	B9 3/4 CJNE R1,#d8,rel8	BA 3/4 CJNE R2,#d8,rel8	BB 3/4 CJNE R3,#d8,rel8	BC 3/4 CJNE R4,#d8,rel8	BD 3/4 CJNE R5,#d8,rel8	BE 3/4 CJNE R6,#d8,rel8	BF 3/4 CJNE R7,#d8,rel8
C8 1/1 XCH A,R0	C9 1/1 XCH A,R1	CA 1/1 XCH A,R2	CB 1/1 XCH A,R3	CC 1/1 XCH A,R4	CD 1/1 XCH A,R5	CE 1/1 XCH A,R6	CF 1/1 XCH A,R7
D8 2/3 DJNZ R0,rel8	D9 2/3 DJNZ R1,rel8	DA 2/3 DJNZ R2,rel8	DB 2/3 DJNZ R3,rel8	DC 2/3 DJNZ R4,rel8	DD 2/3 DJNZ R5,rel8	DE 2/3 DJNZ R6,rel8	DF 2/3 DJNZ R7,rel8
E8 1/1 MOV A,R0	E9 1/1 MOV A,R1	EA 1/1 MOV A,R2	EB 1/1 MOV A,R3	EC 1/1 MOV A,R4	ED 1/1 MOV A,R5	EE 1/1 MOV A,R6	EF 1/1 MOV A,R7
F8 1/1 MOV R0,A	F9 1/1 MOV R1,A	FA 1/1 MOV R2,A	FB 1/1 MOV R3,A	FC 1/1 MOV R4,A	FD 1/1 MOV R5,A	FE 1/1 MOV R6,A	FF 1/1 MOV R7,A

4.5 Examples of MSC121x Instructions

For a particular application, suppose it is required to compute the logical function:

Q = (W & X) + Y + not(Z)

given a byte where Q is bit 7 of port 2, W is bit 0, X is bit 1, Y is bit 2 and Z is bit 3.

The assembler listing below shows how this can be achieved in a number of different ways and allows the reader to see the application of many different types of instructions.

Example 4−3.Assembler Code

```
; Assembly Language Example
$include (reg1210.inc)
W bit ACC.0
X bit ACC.1
Y bit ACC.2
Z bit ACC.3
Q bit P2.7

 CSEG AT 0100H

main:      mov R7,#0 ;initial value
main_1:    lcall fun1             ; decision tree
           lcall fun2             ; 'better' tree ?
           lcall fun3             ; boolean operations
           lcall fun4             ; look-up table
           lcall fun5             ; faster
           lcall fun6             ; fastest
           inc R7
           cjne R7,#10H,main_1    ; try values 00 to 0F
           sjmp main
```

Example 4−3. (Continued)

```
;Max Clocks:(MSC121x 120) (8051 204) Ratio=1.7
fun1:       mov A,R7               ; get input values W, X, Y, Z
            anl A,#8h              ; select' Z
            cjne A,#0,fun1_1       ; test for Z = 0
            sjmp fun1_setQ         ; set Q=1 because Z=0
fun1_1:     mov A,R7               ; recover input values
            anl A,#4               ; select Y
            cjne A,#4,fun1_2       ; test for Y = 1
            sjmp fun1_setQ         ; set Q=1 because Y = 1
fun1_2:     mov A,R7               ; recover input values
            anl A,#3h              ; select W, X
            cjne A,#3,fun1_clrQ    ; test for W = X = 1
            sjmp fun1_setQ
fun1_clrQ:  clr  Q                 ; clear Q
            sjmp fun1_z
fun1_setQ:  setb Q                 ; set Q
fun1_z:     ret

;Max Clocks:(MSC121x 114) (8051 252) Ratio=2.2
fun2:       mov A,R7               ; get input values Z,Y,X,W
            anl A,#8               ; select Z
            jz fun2_setQ           ; set Q because Z = 0
            mov A,R7               ; recover inputs
            anl A,#4               ; select Y
            jnz fun2_setQ          ; set Q because Y = 1
            mov A,R7               ; recover inputs
            rrc A                  ; W into carry
            mov R0,A               ; X in bit #0
            rlc A                  ; recover W
            anl A,R0               ; AND with X
            anl A,#1               ; get just W&X
            jnz fun1_setQ          ; set Q because W&X = 1
            clr  Q                 ; Q=0
            sjmp fun2_z
fun2_setQ:  setb Q                 ; Q=1
fun2_z:     ret
```

Example 4−3. (Continued)

```
;Clocks:(MSC121x 60) (8051 144) Ratio=2.4
fun3:       mov A,R7                ; get input values Z,Y,X,W
            mov C,W                 ; Carry = W
            anl C,X                 ; Carry = W&X
            orl C,Y                 ; Carry = W&X + Y
            orl C,/Z                ; Carry = W&X + Y + /Z
            mov Q,C                 ; Output new Q value
            ret

;Clocks:(MSC121x 64) (8051 120) Ratio=1.9
fun4:       mov A,R7                ; get input values Z,Y,X,W
            anl A,#0FH              ; ensure just 4 bits
            add A,#(fun4_t-fun4_1) ; offest for instructions
            movc A,@A+PC            ; get table entry
fun4_1:     mov C,ACC.0             ; lsb into carry
            mov Q,C                 ; and hence Q
            ret
fun4_t:     db 1,1,1,1,1,1,1,1      ; table represents easy way
            db 0,0,0,1,1,1,1,1      ; to implement any function

;Clocks:(MSC121x 52) (8051 108) Ratio=2.1
fun5:       mov A,R7                ; get input values Z,Y,X,W
            xrl a,#08H              ; complement Z
            clr C                   ; clear carry
            subb A,#3               ; test for boundary
            cpl C                   ; correct polarity
            mov Q,C                 ; and output to Q
            ret

;Clocks:(MSC121x 44) (8051 84) Ratio=1.9
fun6:       mov A,R7                ; get input values Z,Y,X,W
            xrl A,#08H              ; complement Z
            add A,#0FDH             ; identify boundary
            mov Q,C                 ; and output to Q
            ret

            end
```

Chapter 5

System Clocks, Timers, and Functions

This chapter describes the system clocks, timers, and functions of the MSC121x.

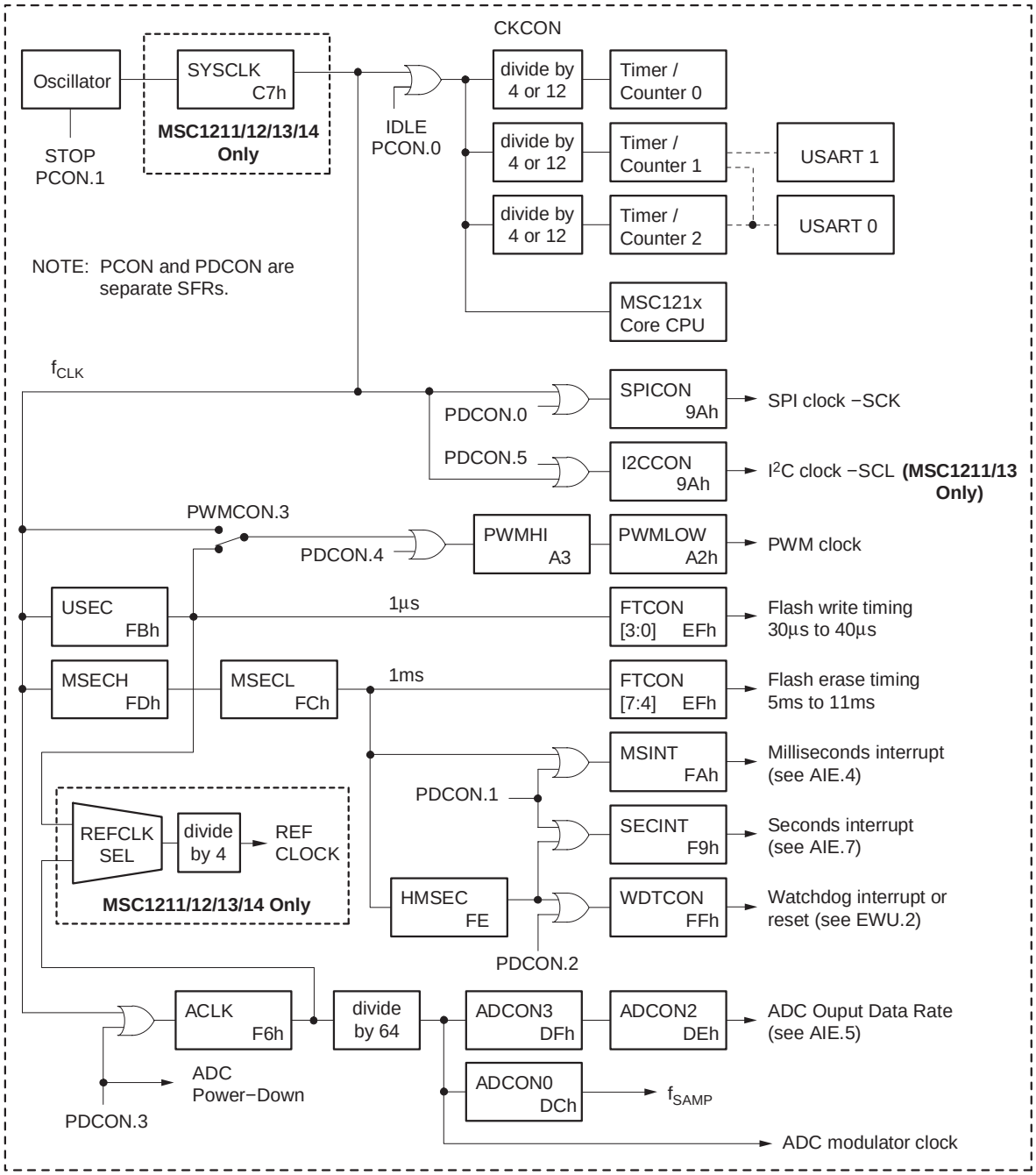
Topic **Page**

5.1	Timing Chain and Clock Controls	5-2
5.2	System Clock Divider	5-5
5.3	Watchdog Timer	5-6
5.4	Low-Voltage Detection	5-8
5.5	Hardware Configuration	5-9
5.6	Breakpoints	5-11

5.1 Timing Chain and Clock Controls

Along with Timer/Counters 0, 1, and 2 found in the 8051/8052 architecture, the MSC121x has numerous additional system timers and clock generators. The main (crystal) oscillator provides the system clock at frequency f_{CLK} either directly, or via a programmable system clock divider. ($t_{CLK} = 1/f_{CLK}$).

Figure 5−1. MSC121x Timing Chain and Clock Control



At power-on, or after a RESET, the signal from the oscillator is not allowed to propagate until after ($2^{17} - 1$) periods. This period of time allows the power rails and crystal oscillator to stabilize. Thereafter, if neither $\overline{\text{PSEN}}$ nor ALE is low, the CPU will begin to execute code starting at location 0000h. While operating, the CPU may set bit 1 of PCON at 87h to assert a STOP condition that can only be exited by a hardware reset. In this condition, all dynamic activity ceases but the port I/O pins retain their levels. To pause the CPU and core peripherals temporarily, bit 0 may be set. This will invoke an IDLE state that is terminated by an auxiliary interrupt associated with AIE at A6h, a wake-up via EWU at C6h, or a RESET. See chapter 13 for further detail on interrupts and their sources. $\overline{\text{PSEN}}$ and ALE are used with RESET to enter serial or parallel flash programming modes.

Subsystems are enabled/disabled by bits in PDCON at F1h in any combination, except that SPI and I^2C subsystems must not be simultaneously active. When a bit is high, the associated subsystems are inactive and power is reduced to a minimum static level.

SPICON [7:5] at 9Ah provides a 3-bit code, *N*, which selects a tap into a binary divider chain to provide the clock for the SPI interface at a frequency of $f_{CLK}/2^{(N+1)}$.

When the I^2C subsystem is active and bit 2 of I2CCON at 9Ah is set, the MSC1211 is in Master mode. The frequency for the I^2C clock is then f_{CLK}/2/(I2CSTAT[7:0] + 1).

The external clock input or crystal oscillator provides the system clock either directly, or via a programmable divider (MSC1211/12/13/14 only). With a system clock of *f* MHz, the program must write (*f* − 1) to the USEC register at FBh in order to provide a clock period as close to 1µs as possible. This clock provides the start and stop timing for the I^2C interface, and is used in conjunction with FTCON [3:0] at EFh to define the Flash memory write cycle timing. The least significant four bits of FTCON are referred to as FWR, and should be set so that $(1 + FWR) \times (USEC + 1) \times 5 \times t_{CLK}$ is between 30µs and 40µs. The designer should consider the relative trade-offs between crystal frequency and accuracy of baud rate generation versus accuracy of other real-time counters.

By default, the output of the USEC divider is used to clock the PWM generator, but f_{CLK} may be selected by setting bit 3 of PWMCON at A1h.The operation of the PWM generator is described later.

Just as USEC is programmed to provide a 1µs reference, MSECH at FDh and MSECL at FCh are used together to provide a signal with a period of 1ms to clock other counters. The period is $(256 \times MSECH + MSECL + 1) \times t_{CLK}$, which may not be an integer number of milliseconds. For example, with a 11.0592MHz crystal and MSECH:MSECL set to 11058, the period will be 1.000018ms. The default value for MSECH:MSECL is 3999_{10} and assumes a 4MHz oscillator. Note that if the system divider, defined by SYSCLK at C7h, is present and active, an extra division factor may be present as well.

The output of MSECH:MSECL clocks three different counters with reload limits set by FTCON [7:4] at EFh, MSINT [6:0] at FAh, and HMSEC [7:0] at FEh.

They define the Flash memory erase timing between 5ms and 11ms, the number of counts for the milliseconds interrupt and the hundreds of millisecond interrupt, respectively. Each counter repeats in *N* + 1 clocks, where *N* is the value written to the bits in each SFR. If bit 7 of MSINT is set, the associated counter will be reloaded as the SFR is written; otherwise, the new value will be loaded next time the count expires.

The interrupt associated with SECINT [6:0] at F9h can be set between 1 and 128 counts of the hundred millisecond counter. If bit 7 of SECINT is set, the associated counter will be reloaded as the SFR is written; otherwise, the new value will be loaded next time the count expires.

The frequency of the ADC modulator is given by:

$$f_{MOD} = \frac{f_{CLK}}{(ACLK + 1) \times 64}$$

where ACLK is the SFR at F6h.

The conversion data rate is given by:

$$\text{ADC Update Rate} = \frac{f_{MOD}}{\text{Decimation Ratio}}$$

The decimation ratio is ADCON3[2:0] at DFh concatenated with ADCON2[7:0] at DEh plus 1, and the ADC output data rate is f_{MOD}/(decimation ratio).

5.2 System Clock Divider (MSC1211/12/13/14)

In order to reduce the average operating power of the microcontroller, a programmable system divider may lower the frequency of the internal clocks.

Table 5−1. SYSCLK—System Clock Divider Register

SYSCLK		SFR C7h	Reset Value = 00h
Bit #	**Name**	**Action or Interpretation**	
7−6	0	Always 0	
5−4	DIVMOD	Clock Divide Mode Write: 00: Normal mode (default, no divide); 01: Immediate mode: start divide immediately; return to Normal mode on an IDLE wakeup condition or direct write to SFR. 10: Delay mode: same as Immediate mode, except that the mode changes with the millisecond interrupt (MSINT). If MSINT is enable, the divide will start on the next MSINT and return to normal mode on the following MSINT. If MSINT is not enabled, the divide will start on the next MSINT condition (even if masked) but will not leave the divide mode until the MSINT counter overflows, which follows a wakeup condition. 11: Medium mode: same as Immediate mode but cannot return to Normal mode on IDLE wakeup condition. Must write directly to SFR. Read: Status 00: No divide 01: Divider is in Immediate mode 10: Divider is in Delay mode 11: Medium mode	
3	0	Always 0	
2−0	DIV	Divide Mode ($f_{CLK} = f_{OSC}$/Divisor) 000: divide-by-2 (default) 001: divide-by-4 010: divide-by-8 011: divide-by-16 100: divide-by-32 101: divide-by-1024 110: divide-by-2048 111: divide-by-4096	

5.2.1 Behavior in Delay Mode 10_2

Changes in the divisor are synchronized with the timeout of the milliseconds system timer, MSINT at FAh, which must be powered up (that is, bit 1 of PDCON at F1h must be 0). Once a new divisor is written to SYSCLK with this mode, it will take effect at the next MSINT timeout. During this time, bit 0 of PCON at 87h can be set to place the CPU in the IDLE state and reduce the power still further.

When the divisor is active and the milliseconds interrupt is enabled via EMSEC (bit 4 of AIE at A6h) the timeout causes immediate removal of the divisor. This is likely to occur when a real-time (elapsed) clock is supported in software by maintaining a record of the accumulated number of millisecond interrupts. The program must compensate for the increase in time caused by the divisor.

In effect, if the milliseconds interrupt is enabled via EMSEC when the divider mode is changed to 10_2, the divisor will become active on the next MSINT interrupt, and return to divide-by-1 on the following MSINT interrupt. However, if the milliseconds interrupt is masked, the divisor will still become active on the next MSINT interrupt, but will not return to divide-by-1 until the milliseconds interrupt after a wake-up condition. If the wake-up condition is caused by an enabled seconds interrupt that is synchronous with a millisecond interrupt, the divider immediately returns to divide-by-1.

5.3 Watchdog Timer

WDTCON [4:0] at FFh plus 1 defines the number of 100ms intervals before the watchdog timer expires, assuming that the watchdog restart sequence is not performed. The watchdog is enabled (or disabled) by writing a 1,0 sequence to bit 7 (or bit 6) of WDTCON. Writing 1,0 to bit 5 restarts the time out.

When the watchdog is enabled and expires, it generates either an interrupt or a reset (default), as determined by bit 3 of HCR0.

WDTI must be cleared within the interrupt service routine. Setting WDTI in software generates a watchdog timer interrupt, if enabled.

Table 5−2. Watchdog Control Bits

Watchdog Interrupt has priority 12 (Low) and jumps to address 63h				
Bit Name	**Abbreviation**	**Name of related SFR**	**Abbreviation**	**Address (Hex)**
Global Interrupt Enable	EA	Interrupt Enable	IE.7	A8
Enable Watchdog Interrupt	EWDI	Extended Interrupt Enable	EIE.4	E8
Watchdog Timer Interrupt flag	WDTI	Enable Interrupt Control	EICON.3	D8
Watchdog Interrupt Priority	PWDI	Extended Interrupt Priority	EIP.4	F8

5.3.1 Watchdog Timer Example Program

When the program is run, it first requires a carriage return (CR) character to be received so that the baud rate can be determined. Thereafter, a CR code must be repeatedly received within 3 seconds; otherwise, the MSC121x is reset and the autobaud routine is restarted.

In Example 5−1, EWDR, bit 3 of HCR0, must be 1 (default) for a reset to occur. In another application, the programmer may clear EWDR so that when the timer expires, an interrupt is requested via WDTI, bit 3 of EICON at D8h.

Example 5−1. Watchdog Timer Program

```
// File WDT.c - Watch Dog Timer
// MSC1210 EVM Switches 1:On  SW3-12345678 SW6-12345678
//                      0:Off     11110111     11110000
#include <Reg1210.h>
#include <stdio.h>
#define xtal 11059200
sbit RedLed    = P3^4;     // RED LED on EVM
sbit YellowLed = P3^5;     // Yellow LED on EVM
code at 0xFFF3 void autobaud(void);

data unsigned char i='A';

void main(void)
{ PDCON&=~0x04;            // power up WatchDog
  MSEC=xtal/1000-1;        // 1ms tick
  HMSEC=100-1;             // 100ms tick
  RedLed=0;                // Turn Red LED on
  autobaud();              // Requires CR
  printf("\nMSC1210 Watchdog Test");
  printf("\nRepeatedly press CR/Enter within 3 seconds\n");
  RI_0 = 0;                // clear received flag in USART
  WDTCON=0x80;             // start watchdog and define
  WDTCON=30;               // 30 * 100ms   timeout
  RedLed=1;                // Turn Red LED off
  while(1){
    while(!RI_0);          // wait for key press
    YellowLed=!YellowLed;  // Toggle Yellow Led
    putchar(i);
    i=(i+1) & 0x5F;             // 32 character sequence
    if((SBUF0 & 0x7F)==0x0D) {  // Test for CR
      WDTCON|=0x20;             // restart Watchdog timer
      WDTCON&=~0x20;       // with 1-0 sequence in bit #5
      }
    RI_0 = 0;              // clear received flag in USART
    }
}
```

5.4 Low-Voltage Detection

Bits 3 and 2 of HCR1 are used to enable a low voltage on either the analog or digital supplies, respectively, to cause a reset. The user may also configure additional low voltages to generate interrupts via LVDCON at E7h.

When high, ALVD or DLVD indicate an active interrupt, while a low level indicates an inactive or masked interrupt.

Table 5−3. LVDCON—Low Voltage Detect Control

LVDCON				**SFR E7h** **Reset Value = 00h**
ALVDIS Bit 7	ALVD2 Bit 6	ALVD1 Bit 5	ALVD0 Bit 4	Analog Threshold of AV_{DD}
DLVDIS Bit 3	DLVD2 Bit 2	DLVD1 Bit 1	DLVD0 Bit 0	Digital Threshold of DV_{DD}
1	x	x	x	Detection Disabled
0	0	0	0	2.7 V (default)
0	0	0	1	3.0 V
0	0	1	0	3.3V
0	0	1	1	4.0 V
0	1	0	0	4.2 V
0	1	0	1	4.5 V
0	1	1	0	4.7 V
0	1	1	1	Analog (pin AIN7) or digital (AIN6) compared with 1.2 V

Table 5−4. Low Voltage Detect

Low-Voltage interrupts have priority 0 (High) and jump to address 33h (shared with other interrupts)				
Bit Name	**Abbreviation**	**Name of related SFR**	**Abbreviation**	**Address (Hex)**
Enable Auxiliary Interrupt	EAI	Enable Interrupt Control	EICON.5	D8
Enable Analog Low-Voltage interrupt	EALV	Auxiliary Interrupt Enable	AIE.1	A6
Enable Digital Low-Voltage Interrupt or Breakpoint interrupt	EDLVB	Auxiliary Interrupt Enable	AIE.0	A6
Auxiliary Interrupt flag	AI	Enable Interrupt Control	EICON.4	D8
Analog Low-Voltage Detect interrupt status flag	ALVD	Auxiliary Interrupt Status Register	AISTAT.1	A7
Digital Low-Voltage Detect or Breakpoint interrupt status flag	DLVD	Auxiliary Interrupt Status Register	AISTAT.0	A7
$= 0010_2$ for analog low voltage $= 0001_2$ for digital low voltage	PAI3−0	Pending Auxiliary Interrupt	PAI.3 to PAI.0	A5

5.5 Hardware Configuration

There are two hardware configuration registers (HCR0 at 7Fh and HCR1 at 7Eh), which form part of 128 bytes of configuration Flash memory. They cannot be accessed directly, as they are not Special Function Registers. Instead, either may be read by first writing its address to CADDR at 93h and then reading CDATA at 94h. Writing to HCR0 or HCR1 can only occur during serial or parallel device programming, when they are mapped to code space addresses 807Fh and 807Eh, respectively.

Table 5−5. Hardware Configuration Register 0

HCR0		**Non-SFR address 7Fh accessed indirectly via SFR CADDR at 93h; Erased Value = FFh**
Bit #	**Name**	**Action or Interpretation**
7	EPMA	Enable Programming Memory Access (security bit). 0: After a reset following programming mode, Flash memory can only be accessed in User Application Mode (UAM) or mass erased. 1: Fully accessible (default)
6	PML	Program Memory Lock (PML has priority over RSL, if RSL = 0). 0: Enable writing to program memory in UAM. 1: Disable writing to program memory in UAM. (default).
5	RSL	Reset Sector Lock. 4 KB of Flash memory from 0000h to 0FFFh. 0: Enable reset sector writing 1: Disable reset sector writing (default)
4	EBR	Enable Boot Rom. 2 KB of read-only memory from F800h to FFFFh. 0: Disable Internal Boot ROM 1: Enable Internal Boot ROM (default)
3	EWDR	Enable Watchdog Reset 0: Disable Watchdog from causing a reset and allow an interrupt if unmasked. 1: Enable Watchdog Reset (default)
2	DFSEL2	DFSEL Data Flash Memory Size. On-chip Flash memory can be partitioned between data memory and program memory. The total memory available depends on the Y version of the device. See section 2.2 for a complete description of memory partitioning.
1	DFSEL1	000: Reserved 001: 4 KB, 8 KB, 16 KB, or 32 KB 010: 4 KB, 8 KB, or 16 KB 011: 4 KB or 8 KB
0	DFSEL0	100: 4 KB 101: 2 KB 110: 1 KB 111: 0 KB

Table 5−6. Hardware Configuration Register 1

HCR1		Non-SFR address 7Eh accessed indirectly via SFR CADDR at 93h; Erased Value = FFh
Bit#	**Name**	**Action or Interpretation**
7	DBLSEL1	Digital Brownout Level Select The digital brownout level is loaded after POR; therefore, a proper POR must occur for digital brownout levels to be properly loaded.
6	DBLSEL0	00: 4.5 V 01: 4.2 V 10: 2.7 V 11: 2.5 V (default)
5	ABLSEL1	Analog Brownout Level Select The analog brownout level is loaded after POR; therefore, a proper POR must occur for analog brownout levels to be properly loaded.
4	ABLSEL0	00: 4.5 V 01: 4.2 V 10: 2.7 V 11: 2.5 V (default)
3	DAB	Disable Analog Power-Supply Brownout Detection 0: Analog Brownout causes reset 1: Analog Brownout reset is disabled (default).
2	DDB	Disable Digital Power-Supply Brownout Detection 0: Digital Brownout causes reset 1: Digital Brownout reset is disabled (default)
1	EGP0	Enable General-Purpose I/O for Port 0 0: Port 0 is used for external memory, P3.6 and P3.7 used for $\overline{WR}$ and $\overline{RD}$. 1: Port 0 is used as general−purpose I/O (default).
0	EGP23	Enable General-Purpose I/O for Ports 2 and 3 0: Port 2 is used for external memory, P3.6 and P3.7 used for $\overline{WR}$ and $\overline{RD}$. 1: Port 2 and Port 3 are used as general-purpose I/O (default)

5.6 Breakpoints

The MSC121x supports hardware breakpoints at addresses in either external data space or code space. When a memory access occurs with an address that matches the value in either of two 16-bit breakpoint registers, an interrupt is generated.

The breakpoint registers can aid system debugging, but caution is needed because of interrupt latency and instruction prefetch. Latency may cause two or three instruction cycles to occur after an address match, while prefetch may trigger a false interrupt; for example, when the breakpoint is placed after a conditional branch that is taken.

Table 5−7. MCON—Memory Control

MCON		SFR 95h Reset Value = 00h
Bit #	**Name**	**Action or Interpretation**
7	BPSEL	Write: 0: select breakpoint register 0 1: select breakpoint register 1 Read: the breakpoint register that created the last interrupt, 0 or 1.
6−5	0	Always 0
4−1	−	Undefined
0	RAMMAP	Write: 0: addresses 0000h to 03FFh in external data memory are on-chip RAM (default) 1: addresses 8400h to 87FFh in external data memory and program memory share the same on-chip RAM

Table 5−8. BPCON—Breakpoint Control

BPCON		SFR A9h Reset Value = 00h
Bit #	**Name**	**Action or Interpretation**
7	BP	Write: 0: No effect 1: clear breakpoint interrupt flag for breakpoint register selected by MCON.7 Read: 0: no breakpoint interrupt 1: breakpoint match from either breakpoint register
6−2	0	Always 0
1	PMSEL	Write: 0: break on address in external data memory 1: break on address in program memory. Applies to breakpoint register selected by MCON.7.
0	EBP	Write: 0: disable interrupt on address match 1: enable interrupt on address match. Applies to breakpoint register selected by MCON.7.

Table 5−9. BPL—Breakpoint Low Address for BP Register Selected in MCON at 95h

BPL		SFR AAh Reset Value = 00h
Bit #	**Name**	**Action or interpretation**
7−0	BPL	Write/Read: Low eight bits of 16-bit breakpoint register. Applies to register selected by MCON.7.

Table 5−10. BPH—Breakpoint High Address for BP Register Selected in MCON at 95h

BPH		SFR ABh Reset Value = 00h
Bit #	**Name**	**Action or interpretation**
7−0	BPH	Write/Read: High eight bits of 16-bit breakpoint register. Applies to register selected by MCON.7.

Table 5−11. Breakpoints

Breakpoint interrupt has priority 0 (high) and jumps to address 33h (shared with DV_{DD} low-voltage interrupt)				
Bit Name	**Abbreviation**	**Name of related SFR**	**Abbreviation**	**Address (Hex)**
Enable Auxiliary Interrupt	EAI	Enable Interrupt Control	EICON.5	D8
Auxiliary Interrupt flag	AI	Enable Interrupt Control	EICON.4	D8
Enable Digital Low Voltage interrupt or Breakpoint interrupt	EDLVB	Auxiliary Interrupt Enable	AIE.0	A6
Digital Low Voltage Detect or Breakpoint interrupt status flag	DLVD	Auxiliary Interrupt Status Register	AISTAT.0	A7

The BP bit in BPCON must be set within the interrupt service routine to clear the interrupt, and bit BPSEL in MCON may be read to determine which breakpoint register caused the interrupt.

Chapter 6

Analog-To-Digital Converters

This chapter describes the digital-to-analog converters (ADCs) of the MSC121x.

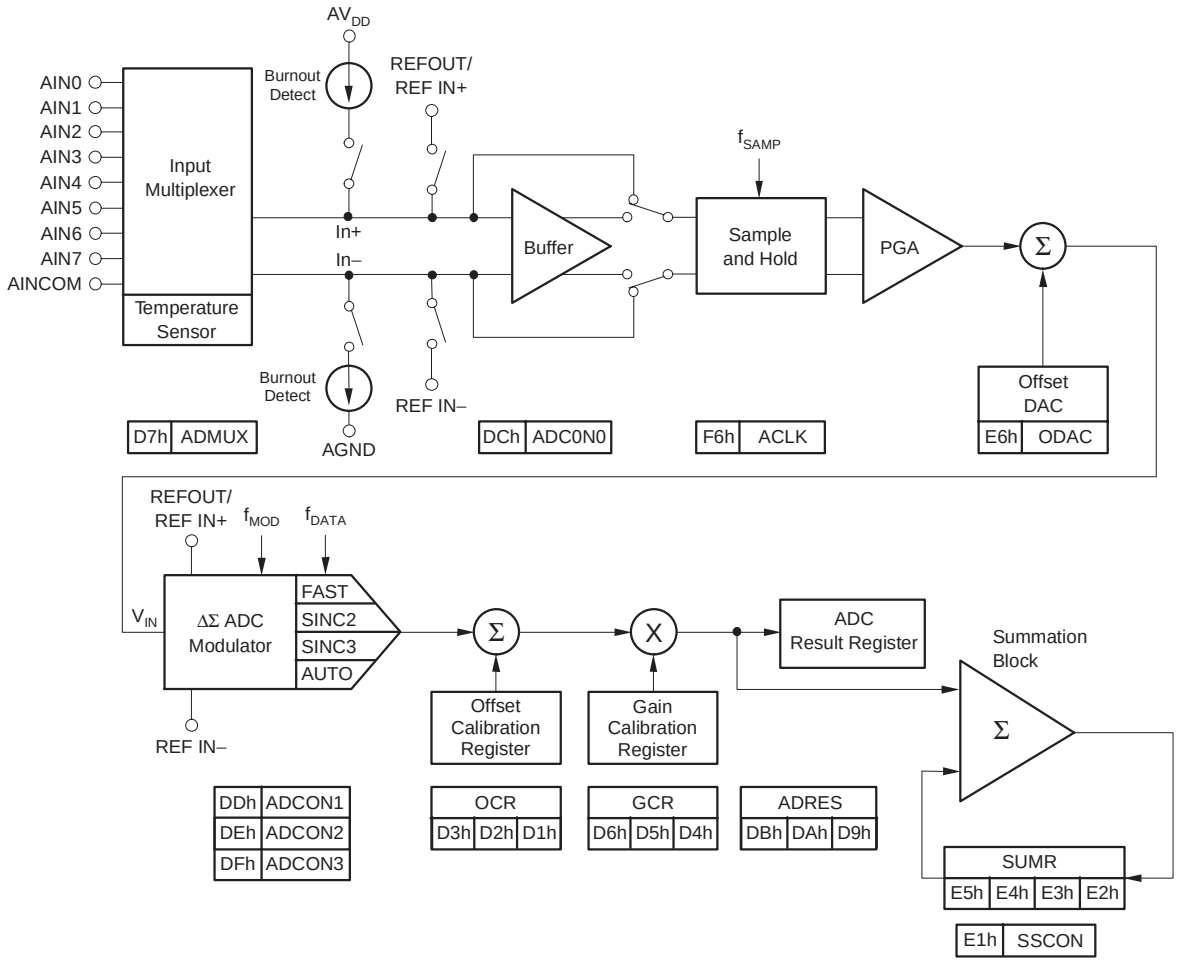
Topic **Page**

6.1	ADC Functional Blocks	6-2
6.2	ADC Signal Flow and General Description	6-3
6.3	Analog Input Stage	6-3
6.4	Input Impedance, PGA, and Voltage References	6-5
6.5	Offset DAC	6-8
6.6	ADC Data Rate, Filters, and Calibration	6-9
6.7	32-Bit Summation Register	6-13
6.8	Accessing ADC Multi-Byte Conversions in C	6-15
6.9	ADC Example Program	6-17

6.1 ADC Functional Blocks

A key feature of the MSC121x that differentiates it from other mixed-signal microcontrollers is a high-precision analog-to-digital subsystem, with a performance that is usually found only in embedded systems with a separate ADC and microprocessor. The major elements of the ADC subsystem are shown in Figure 6−1.

Figure 6−1. ADC Subsystem Elements



6.2 ADC Signal Flow and General Description

Analog signals from pins AIN0 to AIN7, AINCOM, or internal temperature-sensitive diodes are selected independently by two analog multiplexers to provide a differential signal to the programmable gain amplifier (PGA), which may optionally be preceded by a high-impedance buffer. An analog offset of up to ±50% of the full range may be injected into the PGA by the Offset DAC.

The delta-sigma (ΔΣ) ADC can be configured for sampling rate and decimation ratio as well as filter type before its output is passed to digital offset and gain calibration stages to give a 24-bit unipolar or bipolar result.

ADC conversions can be automatically added to a 32-bit summation register (SUMR3 to SUMR0), which is considerably more efficient than using machine-code instructions. A defined number of conversions may also trigger an automatic right shift to produce an averaged value. The CPU can control the 32-bit hardware accumulator directly, as long as the ADC subsystem is powered up. All MSC121x family parts except for the MSC1210 also support 32-bit subtraction.

6.3 Analog Input Stage

Special function register ADMUX at D7h provides two groups of four bits each that specify the analog source channels for the noninverting (positive) and inverting (negative) inputs to the buffer and/or the PGA.

The upper four bits control the noninverting input while the lower four bits control the inverting input. Codes 0000_2 to 0111_2 represent channels AIN0 to AIN7, respectively. Code 1000_2 selects AINCOM, and if both codes are 1111_2, two temperature-sensitive diodes are selected.

When Burnout Detection is enabled, current sources cause the inputs to be pulled to either AV_{DD} or AGND if the selected channel is open circuit, as may happen when a resistive sensor is broken.

The internal diodes are used to provide a temperature-sensitive differential voltage of approximately:

$$V = \frac{nk\ \ln(80)}{q}(T_C + 273.16) = \alpha T_C + \beta$$

For typical values of α and β, refer to the respective data sheets.

For further information about accuracy and calibration, see Texas Instruments application report SBAA100, *Using the MSC121x as a High-Precision Intelligent Temperature Sensor*, available for download at www.ti.com.

Figure 6−2. Input Multiplexer Configuration

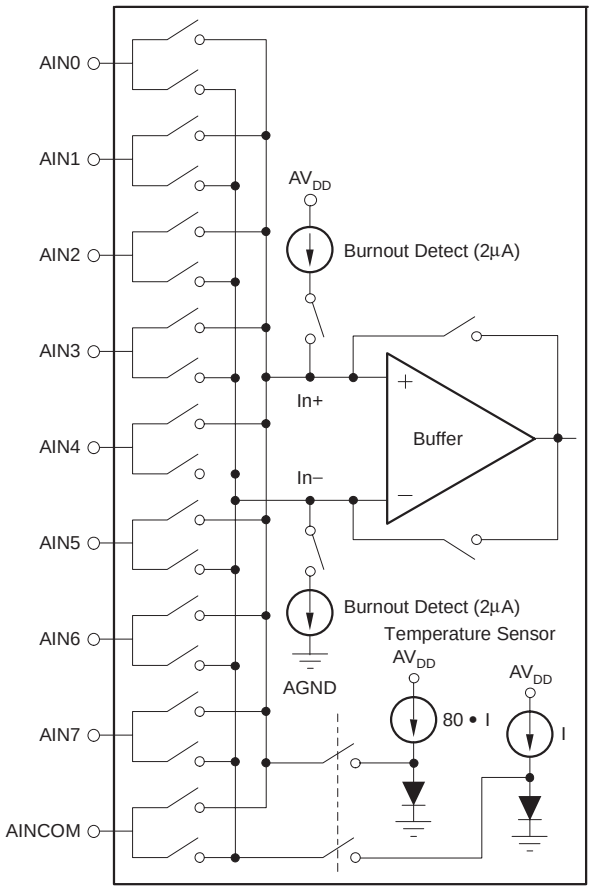


Table 6−1. ADMUX—ADC Multiplexer

ADMUX				SFR D7h — Reset Value = 01h
INP3 Bit 7	INP2 Bit 6	INP1 Bit 5	INP0 Bit 4	Positive input selection
INN3 Bit 3	INN2 Bit 2	INN1 Bit 1	INN0 Bit 0	Negative input selection
0	0	0	0	AIN0 (default positive input)
0	0	0	1	AIN1 (default negative input)
0	0	1	0	AIN2
0	0	1	1	AIN3
0	1	0	0	AIN4
0	1	0	1	AIN5
0	1	1	0	AIN6
0	1	1	1	AIN7
1	0	0	0	AINCOM
1	1	1	1	Temperature sensor. Requires ADMUX = FFh.

6.4 Input Impedance, PGA, and Voltage References

When the buffer is enabled, the input current is typically 0.5nA (impedance is over 1GΩ) and the common-mode range is from AGND + 50mV to AV_{DD} − 1.5 V. The buffer should be enabled whenever burnout detection is used.

However, when the buffer is not enabled, each analog input is presented with a dynamic load such that the mean differential impedance is (7MΩ/*G*); where *G* is defined in Table 6−2. The input impedance is lowered and varies with gain; however, the input range is from AGND − 0.1 V to AV_{DD} + 0.1 V.

Table 6−2. Impedance Divisor (G) for a Given PGA

PGA	1	2	4	8	16	32	64	128
G	1	2	4	8	16	32	64	64

When the buffer is not selected, the input impedance of the analog input changes with ACLK clock frequency (ACLK, SFR F6h) and gain (PGA). The relationship is:

$$A_{IN}\ \text{Impedance}\ (\Omega) = \left(\frac{1\text{MHz}}{\text{ACLK Frequency}}\right) \cdot \left(\frac{7\text{M}\Omega}{\text{G}}\right)$$

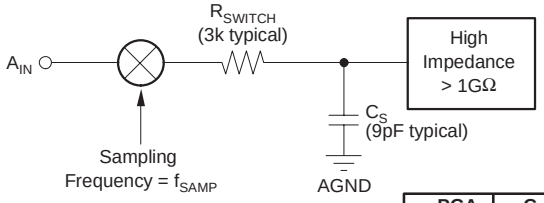
where:

$$\text{ACLK frequency}\ (f_{ACLK}) = \frac{f_{CLK}}{\text{ACLK} + 1}$$

$$f_{MOD} = \frac{f_{ACLK}}{64}.$$

Figure 6−3 shows the basic input structure of the MSC121x.

Figure 6−3. Analog Input Structure without Buffer



PGA	C_S
1	9pF
2	18pF
4 to 128	36pF

PGA	BIPOLAR MODE FULL-SCALE RANGE	UNIPOLAR MODE FULL-SCALE RANGE	f_{SAMP}
1	$\pm V_{REF}$	$+V_{REF}$	f_{MOD}
2	$\pm V_{REF}/2$	$+V_{REF}/2$	f_{MOD}
4	$\pm V_{REF}/4$	$+V_{REF}/4$	f_{MOD}
8	$\pm V_{REF}/8$	$+V_{REF}/8$	$f_{MOD} \cdot 2$
16	$\pm V_{REF}/16$	$+V_{REF}/16$	$f_{MOD} \cdot 4$
32	$\pm V_{REF}/32$	$+V_{REF}/32$	$f_{MOD} \cdot 8$
64	$\pm V_{REF}/64$	$+V_{REF}/64$	$f_{MOD} \cdot 16$
128	$\pm V_{REF}/128$	$+V_{REF}/128$	$f_{MOD} \cdot 16$
NOTE: f_{MOD} = ACLK frequency/64			

Table 6−3. ADCON0—ADC Control Register 0

ADCON0		SFR DCh — Reset Value = 30h
Bit #	**Name**	**Action or Interpretation**
7	REFCLK	Reference Clock (MSC1211/12/13/14 only) The reference is specified with a 250kHz clock. REFCLK should be selected by choosing the appropriate source so that it does not exceed 250kHz. 0 : $\frac{t_{CLK}}{(ACLK + 1) \times 4}$ 1 : $\frac{USEC}{4}$
6	BOD	Burnout Detect When enabled, a 2µA current source is connected from AV_{DD} to the positive input, while a 2µA current sink is connected from the negative input to ground. Write: 0: Burnout Current Sources Off (default) 1: Burnout Current Sources On
5	EVREF	Enable Internal Voltage Reference Write: 0: Internal Voltage Reference Off 1: Internal Voltage Reference On (default). If the internal voltage reference is not used, it should be turned off to save power and reduce noise.
4	VREFH	Voltage Reference High Select Write: 0: REFOUT is 1.25 V 1: REFOUT is 2.5 V (default)
3	EBUF	Enable Buffer Write: 0: Buffer disabled (default) 1: Buffer enabled, results in increased power and impedance but reduced range.
2−0	PGA	Programmable Gain Amplifier Write: 000 to 111: Gives a gain G = 2^{PGA} or 1 (default) to 128

Bits of ADCON0 determine various parameters according to Table 6−4.

By default, the internal voltage reference is turned on at 2.5V, when the ADC subsystem is powered up. Therefore, if an external reference is provided, the internal reference should be disabled via EVREF before bit 3 of PDCON at F1h is cleared.

If the internal voltage reference is to be used, the default level of 2.5V is allowed only if AV_{DD} is between 3.3V and 5.25V. The internal 1.25V V_{REF} can be used over the entire analog supply range (AV_{DD} = 2.7V to 5.25V).

When the internal voltage reference is disabled, an external differential reference is represented by the voltage between REF IN+ and REF IN−. This permits ratiometric measurements, but the absolute voltage on either input must be from AGND to AV_{DD}.

In both cases, the REF IN+ pin should have a 0.1µF capacitor to AGND.

Table 6−4. ADCON0 Bit Parameters

PGA Bits [2:0]	Gain	Full-Scale Range	Sampling Frequency	Effective Number of Bits at 10Hz Rate	RMS Resolution for V_{REF} = 2.5V (nV)
000	1	$\pm V_{REF}$	f_{MOD}	21.7	1468
001	2	$\pm V_{REF}/2$	f_{MOD}	21.5	843
010	4	$\pm V_{REF}/4$	f_{MOD}	21.4	452
011	8	$\pm V_{REF}/8$	2 f_{MOD}	21.2	259
100	16	$\pm V_{REF}/16$	4 f_{MOD}	20.8	171
101	32	$\pm V_{REF}/32$	8 f_{MOD}	20.4	113
110	64	$\pm V_{REF}/64$	16 f_{MOD}	20	74.5
111	128	$\pm V_{REF}/128$	16 f_{MOD}	19	74.5

6.5 Offset DAC

The input to the PGA may be offset by up to ±50% of the range via the offset DAC. This 8-bit DAC is controlled by ODAC at E6h with a coding scheme such that the most significant bit represents the sign of the offset, while the least significant seven bits represent the magnitude. When the magnitude is zero, the ODAC is disabled and the voltage into the PGA is not offset.

$$\text{Offset} = \frac{V_{REF}}{2^{PGA}}\left(\frac{\text{ODAC}[6:0]}{127}\right)(-1)^{[\text{ODAC7}]}$$

where PGA is the gain of the programmable gain amplifier

Here, V_{REF} is the voltage on the REF IN+ pin with respect to REF IN− and should not be confused with the internal voltage reference that is with respect to AGND.

The gain error of the 8-bit ODAC is typically about ±1.5% of its range, which means its absolute accuracy can be significant in some applications. However, it is monotonic with an integral nonlinearity of less than 0.25 bits and has a temperature coefficient of typically 1ppm/°C. It may be used in a predictive manner with due regard to its range, resolution, stability, and accuracy, or it may be calibrated using the ADC.

6.6 ADC Data Rate, Filters, and Calibration

The data rate for ADC conversions is determined by the frequency of the modulator clock, f_{MOD}, and the decimation ratio, which is a right-justified, 11-bit field in ADCON3 at DFh (high) concatenated with ADCON2 at DEh (low). Its default value is 1563.

$$\text{ADC Output Data Rate} = f_{DATA} = \frac{f_{MOD}}{\text{Decimation Ratio}} = \frac{1}{t_{DATA}}$$

$$\text{where } f_{MOD} = \frac{f_{CLK}}{(\text{ACLK} + 1) \times 64}$$

When the decimation ratio, PGA, AV_{DD}, or temperature are changed, the ADC must be recalibrated.

The mode of operation of the ADC is controlled by ADCON1 at DDh. This determines whether the inputs are interpreted as unipolar or bipolar, the type of digital filter, and the type of calibration.

Table 6−5. ADCON1—ADC Control Register 1

ADCON1		**SFR DDh** **Reset Value = 30h**
Bit #	**Name**	**Action or Interpretation**
7	OF_UF	Summation Invalid If this bit is set, the data in the summation register is invalid. Either an overflow or underflow occurred. The bit is cleared by writing a ‘0’ to it.
6	POL	Polarity Write: 0: Bipolar such that −FSR = 0x800000, zero = 0x000000 and +FSR = 0x7FFFFF 1: Unipolar such that −FSR = 0x000000, zero = 0x000000 and +FSR = 0xFFFFFF
5	SM1	Settling Mode Write: 00: Auto 01: Fast 10: $Sinc^2$ 11: $Sinc^3$
4	SM0	
3	—	Not Used
2	CAL2	Calibration Control (number of t_{DATA} periods to complete) Write: 000: No Calibration (default) 001: Self Calibration for Offset and Gain (14) 010: Self Calibration for Offset only (7) 011: Self Calibration for Gain only (7) 100: System Calibration for Offset only (7) 101: System Calibration for Gain only (7) 110: Reserved 111: Reserved Read: 000_2
1	CAL1	
0	CAL0	

When the voltage presented to the ADC changes, the time it takes to receive valid data depends upon the type of filter that is selected, as well as the conversion time, t_{DATA}. Higher-order filters provide better noise immunity but take longer to settle, and the user must make considered judgments as to system performance based on resolution, settling time, and notch frequency.

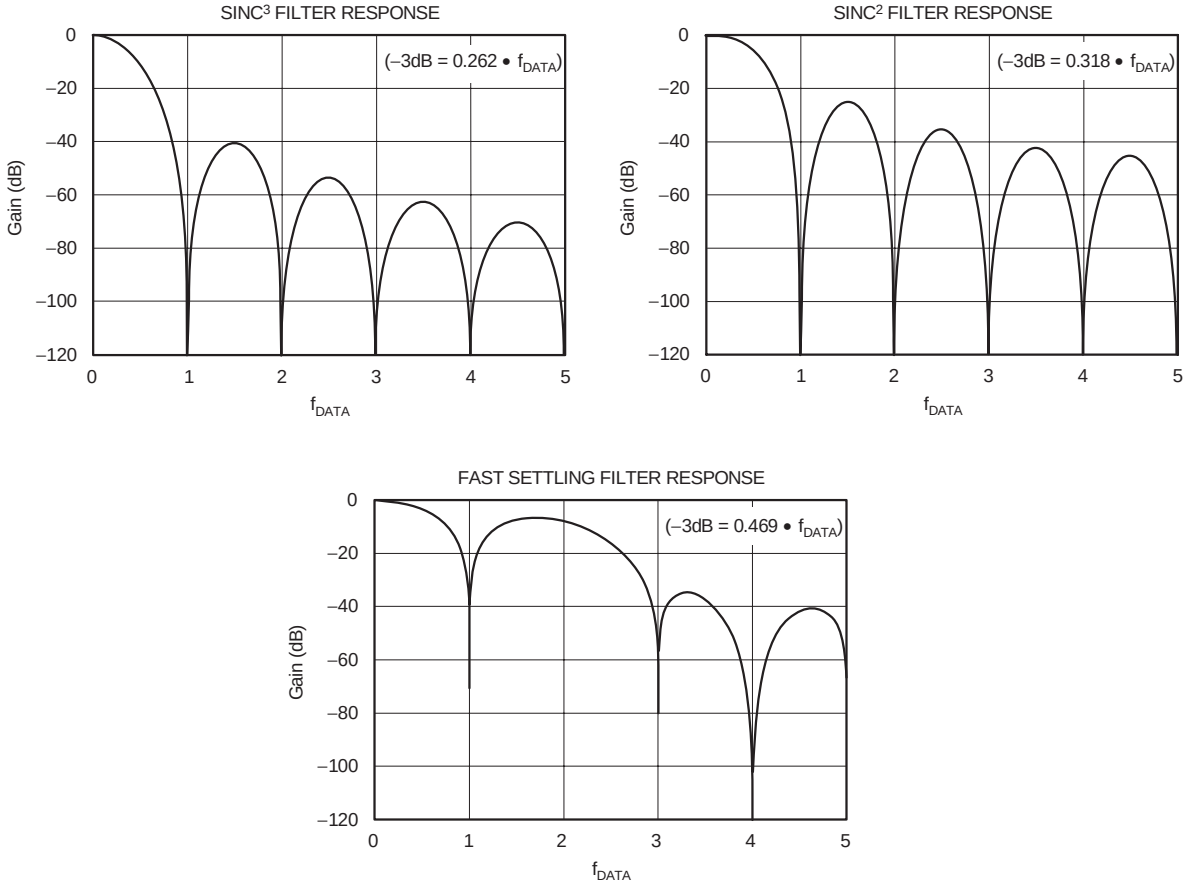
In Auto mode, the type of filter that is used changes whenever the input multiplexer, ADMUX, or PGA are altered. The ADC first makes two conversions using the Fast filter, then one with $Sinc^2$, and then one with $Sinc^3$.

In the graphs shown in Figure 6−4, f_{DATA} = Data Output Rate = 1/ t_{DATA}

The ADC performs conversions at a regular rate of f_{DATA}, as shown in the following equation:

$$f_{DATA} = \left(\frac{f_{CLK}}{64 \times (ACLK + 1) \times \text{ Decimation Ratio}} \right)$$

Figure 6−4. Filter Frequency Responses



NOTE: f_{DATA} = Data Output Rate = $1/t_{DATA}$

In applications where more than one analog input is measured, the program should write different values to ADMUX in a way that is synchronized with conversions to get the best throughput rate. Ideally, ADMUX should be updated as soon as the ADC interrupt flag is set, but there will always be a software delay. Assuming the delay is less than $20 \times t_{MOD}$ and the decimation ratio is large (over 1000), any error introduced is less than intrinsic noise.

The 24-bit result is held in the logically concatenated registers ADRESH (high), ADRESM, and ADRESL (low), at SFR addresses DBh, DAh, and D9h, respectively. These registers are loaded when a conversion is completed, as long as ADRESL has been read since the last value was written.

In devices that have the AIPOL register, reading AIE may return the mask bits that were previously written. Therefore, these devices support read/modify/write instructions such as ORL AIE,#020H to enable the ADC interrupt and ANL AIE,#0BFH to disable the summation interrupt. However, this must not be attempted with other devices where reading AIE returns the value of the interrupt flags before masking. To allow dynamic modification of interrupt enable bits on these parts, the programmer should first manipulate a byte in memory with read/modify/write instructions, and then copy it to AIE. If the memory byte is updated by a sequence of instructions, in general they should be protected from interrupts.

Table 6−6. ADC Interrupt Controls

Family Part	Bit 5 of AIE at A6h Enable ADC Interrupt	Bit 5 of AISTAT at A7h ADC Interrupt Status Flag	Bit 5 of AIPOL at A4h ADC Interrupt Poll
MSC1211 MSC1212 MSC1213 MSC1214	Write: 0: Masked 1: Enabled Read: ADC interrupt flag before masking (RDSEL = 0) or value of EADC (RDSEL = 1).	Read: 0: Inactive or masked 1: Active While active, no new data will be written to ADRES. Cleared by reading ADRESL at D9h.	Read: ADC interrupt flag before masking (RDSEL = 1) or value of EADC (RDSEL = 0).
MSC1210	Write: 0: Masked 1: Enabled Read: Mask value	Read: 0: Inactive or masked 1: Active While active, no new data will be written to ADRES. Cleared by reading ADRESL at D9h.	Not present

When the MSC121x is reset, default values are loaded into the digital offset and digital gain calibration registers associated with the ADC. Specifically, for offset OCH:OCM:OCL = 00000000h and for gain GCH:GCM:GCL = 5FEC5Ah. (See Application Note SBAA099, *Calibration Routines and Register Value Generation for the ADS121x Series*, for additional information.) Although the ADC will then produce an output that varies linearly with the differential input voltage, it will not have the correct scale. A program is able to write any desired value to these calibration SFRs, but is most likely to set the CAL bits in ADCON0 to force an internal calibration for offset and gain (CAL = 001). A differential input of V_{REF} = (REF IN+) − (REF IN−) will then map to a full-scale digital output. Alternatively, the overall system can be placed into a defined zero state and then calibrated for offset (CAL = 100) followed by a full-scale condition and calibrated for gain (CAL = 101). Each type of calibration takes seven t_{DATA} periods, as summarized in Table 6−3. CAL = 001 takes 14 t_{DATA} periods. For best results, calibration should be performed with the $Sinc^3$ or auto filter selected.

6.7 32-Bit Summation Register

To use the 32-bit summation register, either under the control of the CPU and/or the ADC, bit 3 of PDCON at F1h must be ‘0’. Operations are controlled by SSCON at E1h, with data accessed via SUMR3:SUMR0.

Table 6−7. Summation Register

Register Name	Address (Hex)	Read Summation Register	Write Temporary Register
SUMR3	E5	Bits 31 to 24 (most significant)	Bits 31 to 24 (most significant)
SUMR2	E4	Bits 23 to 16	Bits 23 to 16
SUMR1	E3	Bits 15 to 8	Bits 15 to 8
SUMR0	E2	Bits 7 to 0 (least significant)	Bits 7 to 0 (least significant)

Table 6−8. SSCON—Summation/Shift Control

SSCON								SFR E1h Reset Value = 00h
Bit Name and Number								Action or Interpretation where: Read of Summation Register = A Write to Temporary Register = B
SSCON1	SSCON0	SSCNT2	SSCNT1	SSCNT0	SSHF2	SSHF1	SSHF0	
7	6	5	4	3	2	1	0	
0	0	x	x	x	x	x	x	Select CPU summation mode for MSC12x
0	0	0	0	0	0	0	0	Clear Summation register, A = zero(1)
0	0	0	1	0	0	0	0	Change to summation mode (2,3) Next CPU summation on write to SUMR0, A = A + B
0	0	1	0	0	0	0	0	Change to subtraction mode (2,3) Next CPU subtraction on write to SUMR0, A = A − B
1	0	0	0	0	S	S	S	Shift right by (SSS_2 + 1) bits
0	1	C	C	C	0	0	0	Add ADC conversions to Summation register $2^{(CCC+1)}$ times (that is, 2 to 256 times).
1	1	C	C	C	S	S	S	Add ADC conversions to Summation register $2^{(CCC+1)}$ times (that is, 2 to 256 times). Then shift right by (SSS_2 + 1) bits and set the summation complete interrupt flag.

1) For the MSC1210, writing 00h to SSCON clears the 32-bit hardware accumulator and selects CPU controlled summation. For other devices, the 32-bit hardware accumulator is cleared, but the mode is not changed.
2) These operations are not available in the MSC1210.
3) If the polarity bit in ADCON1 at DDh is 0, the 24-bit ADC conversion is sign extended to 32 bits. That is, bit 7 of ADRESH is propagated to all higher bits.

Immediately after a CPU instruction writes data to SUMR0, it may trigger an addition, subtraction, or shift, depending on the value of SSCON. Addition and subtraction take a single cycle, $t_{CLK.}$ Shifting is performed either 1 or 2 bits per cycle, and takes up to 4 t_{CLK} periods to complete.

Table 6−9. Summation Interrupt Controls

Family Part	Bit 6 of AIE at A6h Enable Summation Interrupt	Bit 6 of AISTAT at A7h Summation Interrupt Status Flag	Bit 6 of AIPOL at A4h Summation Interrupt Poll
MSC1211 MSC1212 MSC1213 MSC1214	Write: 0: Masked 1: Enabled Read: Summation interrupt flag before masking (RDSEL = 0) or value of ESUM (RDSEL = 1).	Read: 0: Inactive or masked 1: Active While active, no new data will be written to SUMR. Cleared by reading SUMR0 at E2h.	Read: Summation interrupt flag before masking (RDSEL = 1) or value of ESUM (RDSEL = 0).
MSC1210	Write: 0: Masked 1: Enabled Read: Mask value	Read: 0: Inactive or masked 1: Active While active, no new data will be written to SUMR. Cleared by reading SUMR0 at E2h.	Not present

6.8 Accessing the ADC Multi-Byte Conversion in C

ADRESH:ADRESM:ADRESL represent a 24-bit register, while SUMR3:SUMR2:SUMR1:SUMR0 represent a 32-bit register. It is often useful to map both of these to long integers in C, but care should be taken. For example, assuming that the variable *sum* has been declared to be of type “signed long int,” it is tempting to write:

```
sum = SUMR3 << 24 + SUMR2 << 16 + SUMR1 << 8 + SUMR0;
```

However, this produces a pattern-dependent incorrect value because of the (ANSI-defined) 16-bit integer promotion rules within most compilers for the 8051 family.

Changing to:

```
sum = ((unsigned long)SUMR3 << 24) + ((unsigned long)SUMR2 <<
16)

    +  ((unsigned long)SUMR1 << 8)  +  (unsigned long)SUMR0;
```

will produce the expected value, but may take between approximately 800 and 1200 machine cycles, as compilers call run-time libraries to achieve multi-bit shifts. Since the order of additions is not defined in C, it is possible that SUMR0 is accessed first and the ADC interrupt flag is cleared. If other interrupts are present and their service routines take more time to complete than the next conversion, SUMR3, 2, 1 may be overwritten before being used to complete the evaluation of the expression.

Another approach is to define a union to overlay byte-wide variables with a 4-byte long integer.

```
typedef union {

   unsigned long v;

   char va[4];

   struct {char v3,v2,v1,v0;} vs;

   } type_sumv;

type_sumv data s;  //variable s is placed in ‘core’ on-chip
data space
```

Then use:

```
s.vs.v3=SUMR3;

s.vs.v2=SUMR2;

s.vs.v1=SUMR1;

s.vs.v0=SUMR0;    // SUMR0 is accessed last

reading = f(s.v); // some function of the 4-byte variable v.s
```

Alternatively, array elements may be used, but the order of subscripts is reversed.

```
s.va[0]=SUMR3;

s.va[1]=SUMR2;

s.va[2]=SUMR1;

s.va[3]=SUMR0;
```

Although the code needed to access the union may appear clumsy, it maps to simple inline assembly-level MOV instructions that take $3 \times 4 = 12$ machine cycles to execute. In other words, it is approximately 100 times faster than using multiple shifts.

In the next example, the ADC results register is read using an assembly-level program, which makes expressions in C more intuitive. This technique may also be used to read the summation register.

6.9 ADC Example Program

Example 6−1 shows how the ADC may be used in a polled environment with a foreground activity that produces a pseudo-random binary data stream. The number of characters output per line equals the temperature of the MSC121x in degrees Celsius (°C). The main program is written in C and calls the Boot ROM to determine the baud rate and an assembly language function to read the ADC conversion. It is intended for use directly with Texas Instruments' MSC1210-DAQ-EVM or full EVMs with an appropriate value for ACLK.

Example 6−1.ADC Program

```
// Polledadc.c - Pseudo Random Binary Sequence generator with Polled ADC
// MSC1210 EVM Switches 1:On  SW3-12345678 SW6-12345678
//                      0:Off     11110111     11110000
#include <Reg1210.h>
#include <stdio.h>
sbit RedLed = P3^4;                  // RED LED on EVM
sbit YellowLed = P3^5;               // Yellow LED on EVM
code at 0xFFF3 void autobaud(void);
extern signed long bipolar(void);    // reads ADC value

void main(void)
{ data char mask=0x8E, r=1,n,j,x, temp=50, count=255;
  data signed long reading; data int iy; data float y;
//PDCON  = 0x0f7;         // would turn adc on, but turn other subsystems off
//PDCON &=~0b00001000;    // turns on adc and leaves other subsystems unchanged
PDCON &=~0x08;            // turns on adc and leaves other subsystems unchanged
//ACLK = 2;   // ACLK frequency =  1.8432MHz/(2+1)  = 0.6144MHz for MSC1210-DAQEVM
  ACLK = 17;  //                = 11.0592MHz/(17+1) = 0.6144MHz for MSC1210EVM
//ACLK = 35;  //                = 22.1184MHz/(35+1) = 0.6144MHz for MSC1211EVM
  DECIMATION = 1920 ;     // => 200ms per conversion
//ODAC=0;                 // offset DAC is zero after RESET
//ADCON0 = 0b00100000;    // BOD off, Vref on, 1.25V, Buff off, PGA 1
  ADCON0 = 0x20;          // BOD off, Vref on, 1.25V, Buff off, PGA 1
  autobaud();
  printf("MSC121x Random bit generator with polled ADC\n");
  printf("Readings begin in (14+3)*200ms = 3.4 seconds \n");
  ADMUX  = 0xff;          // Select Temperature diodes
  ADCON1 = 0x01;          // bipolar, auto mode, self calibration - offset and gain
```

Example 6−1. (Continued)

```
  for (j=0;j<3;j++) {
    while (!(AIE & 0x020)) {}
    reading=bipolar();                  // discard 3 conversions after calibration
  }
  RI_0 = 0;                             // Clear received flag in USART
  while (!(AIE & 0x20));                // wait for conversion
  while(1){
    while(!RI_0) {
      if (AIE & 0x20) {                 // test ADC interrupt flag
        reading=bipolar();              // get reading and clear flag
//      y=(reading-692199)/2534.1;      // simple theoretical
//      y=(reading-700875)/2567.1;      // convert to Degrees C (empirical 1)
        y=(reading-704509)/2595.1;      // convert to Degrees C (empirical 2)
        iy=y+0.5;                       // nearest integer
        if ((iy>0) && (iy<50)) temp=iy; // clamp range
        }
      if (count>=temp) {                // if line length >= temperature
        printf("\n%3d ",temp);          // output new line and temperature
        YellowLed=RedLed;
        count=0;
        }
      n=r & mask;                       // PRBS generator with 4-bit feedback
      j=0;                              // j will become the sum of 1's in n
      while (n)
        {n&=(n-1); j++;}
      r=(r+r)+(j&1);                    // shift r left with lsb of sum
      if (r&1) putchar('*');            // Note: putchar takes 28 machine cycles
      else     printf(".");             // but   printf takes 354 machine cycles
      count++;                          // increment character count
    }
  RI_0 = 0;
  while(!RI_0);   // wait for character
  RI_0 = 0;
  }               // continue
}
```

Example 6−1. (Continued)

From TI file Utilities.A51

```
File name: utilities.a51
;
; Copyright 2003 Texas Instruments Inc as an unpublished work.
; All Rights Reserved.
;
; Revision History
; Version 1.1
;
; Assembler Version  (Keil V2.38), (Raisonance V6.10.13)
;
; Module Description:
; ADC routines to read 24-bit ADC and return the value as a long integer.

;*********************************************************************
$include (legal.a51) ; Texas Instruments, Inc. copyright and liability
$include (reg1210.inc)
;*********************************************************************
PUBLIC unipolar, bipolar, read_sum_regs
adc_sub SEGMENT  CODE
    RSEG    adc_sub

;;;;;;;;;;;;;;;;;;;;
; unsigned long unipolar(void)
; return the 3 byte adres to R4567 (MSB~LSB)
; unsigned long int with R4=0
unipolar:
    mov    r4,#0
    mov    r5,adresh
    mov    r6,adresm
    mov    r7,adresl
    ret

;;;;;;;;;;;;;;;;;;;;
; signed long bipolar(void)
; return the 3 byte adres to R4567 (MSB~LSB)
; return signed long int with sign extendsion on R4
bipolar:
    mov    r4,#0
    mov    a,adresh
    mov    r5,a
    mov    r6,adresm
    mov    r7,adresl
    jnb    acc.7,positive
    mov    r4,#0ffh
positive:
    ret
```

Example 6−1. (Continued)

```
;;;;;;;;;;;;;;;;;;;;
; signed long read_sum_regs(void)
; return the 4 byte sumr to R4567 (MSB~LSB)
; return signed long int, sign extension done by hardware
read_sum_regs:
    mov    r4, SUMR3;
    mov    r5, SUMR2;
    mov    r6, SUMR1;
    mov    r7, SUMR0;
    ret

    end
```

Produces

```
MSC121x Random bit generator with polled ADC
Readings begin in (14+3)*200ms = 3.4 seconds
 25 .**...****.*....********.
 25 .*....*.*..*****.*.*.*.**
 25 *.....**...*.*.**..**..*.
 25 ******.****..**.***.***..*
 26 .*.*..*.*...*..*.**.*...**
 26 ..***..****...**.**....*...
 27 *.***.*.****.**.*****....**
 27 .*..**.*.**.**.*.*.....*..*
 27 **.**..*..*..**......***.*..
 28 *...***...*.......*.**...***
 28 *.*....********..*....*.*..*
 28 ****.*.*.*.***.....**...*.*.
 28 **..**..*.******.****..**.**
 28 *.***..*.*.*..*.*...*..*.**.
 28 *...**..***..****...**.**....
 29 *...*.***.*.****.**.*****....
 29 **.*..**.*.**.**.*.*.....*..*
```

Note:

The temperature is shown in degrees Celsius (°C) followed by the same number of pseudo-random characters. The temperature was increased from 25°C to 29°C by touching the MSC121x with a finger.

The `.**...****.` pattern repeats every 255 characters.

Chapter 7

Digital-To-Analog Converters

This chapter describes the digital-to-analog converters (DACs) of the MSC121x.

Topic **Page**

7.1	Introduction	7-2
7.2	DAC Selection	7-3
7.3	Configuration and Control	7-5
7.4	DAC Technology and Limitations	7-7
7.5	DAC Example Program	7-8

7.1 Introduction

The MSC1211/12 contain four mutually independent 16-bit DACs, referred to as DAC0 to DAC3. The MSC1213/14 contain two mutually independent 16-bit DACs, referred to as DAC0 to DAC1. Each produces a voltage as shown in the following equation:

$$V_{DAC} = \text{DAC REF} \times \frac{\text{DAC}}{65,536}$$

where:

DAC REF is the selected DAC reference

DAC is the value written to the DAC register.

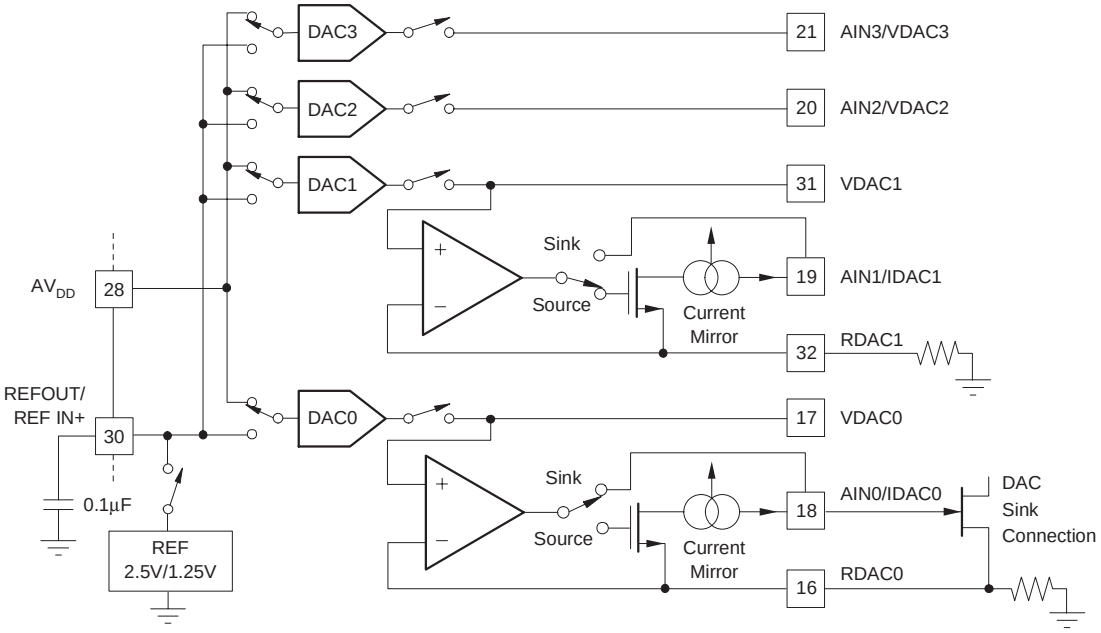
PDDAC, bit 6 of PDCON at F1h, must be ‘0’ for the DACs to be altered via DACL, DACH, and DACSEL at B5h, B6h and B7h, respectively. When PDDAC is ‘1’, a DAC may remain active. To power-down and isolate a DAC output, the output mode bits, DOMx_1 and DOMx_0, in the appropriate control register must both be ‘1’ (default).

When V_{REF} is selected, the voltage on the REFOUT/REF IN+ pin is used as the reference for the DAC. Consequently, if either the 2.5V or 1.25V on-chip reference is used, the ADC subsystem has to be powered up using bit 3 of PDCON.

In addition, voltage-to-current converters may be selectively enabled for DAC0 (or DAC1) and result in a scaled current, as well as a voltage on separate pins. If bit 5 of DAC0CON (or DAC1CON) is ‘0’, a current equal to DAC0/RDAC0 (or DAC1/RDAC1) is generated via a current mirror and flows out of the MSC1211 from the AV_{DD} supply.

The analog pathways are depicted in Figure 7−1, along with pin allocations, some of which are multiplexed with inputs to the ADC.

Figure 7−1. DAC Architecture



7.2 DAC Selection

Each DAC has an 8-bit control register, a buffered 16-bit data register, and two additional bits, which determine the way that the output data register is loaded.

Three SFRs are used to access and control the DACs using an indirect addressing scheme. This configuration makes accessing each DAC more involved than simply writing to independent SFRs, but has the advantage of using the SFR memory space efficiently.

The DAC select register (SFR B7h), determines the individual DAC buffer or control register to be accessed, as well as the shared Load Control Register. The interpretation of SFRs B6h and B5h depends upon the value in DACSEL, as shown in Table 7−1.

Table 7−1. DACSEL Values

DACSEL (B7h)	DACH (B6h)	DACL (B5h)	Reset Value
00h	DAC0(high)	DAC0(low)	0000h
01h	DAC1(high)	DAC1(low)	0000h
02h	DAC2(high)	DAC2(low)	0000h
03h	DAC3(high)	DAC3(low)	0000h
04h	DAC1CON	DAC0CON	6363h
05h	DAC3CON	DAC2CON	0303h
06h	—	LOADCON	xx00h
07h to FFh	Reserved	Reserved	Reserved

The way a DAC output register is loaded is determined by two bits in the Load Control Register (LOADCON) as shown in Table 7−2 and Table 7−3. The LOADCON register is accessed indirectly via the SFR at B5h when DACSEL = 06h.

Table 7−2. LOADCON SFR

DACSEL = 06h	7	6	5	4	3	2	1	0	Reset Value
SFR B5h	DAC3		DAC2		DAC1		DAC0		00h
	D3LOAD1	D3LOAD0	D2LOAD1	D2LOAD0	D1LOAD1	D1LOAD0	D0LOAD1	D0LOAD0	

Table 7−3. DxLOAD Output Modes for DACx

DxLOAD[1:0]	DxLOAD Output Mode for DACx
Direct load 00	A write to the DAC high byte (DACxH) is directed to the upper byte of the 16-bit data buffer and does not alter the output register. A write to the DAC low byte (DACxL) is directed to the lower byte of the 16-bit data buffer, which is immediately copied to the output register.
Delayed load 01	Values are written to the DACxH/DACxL16-bit data buffer, which will be transferred to the DAC output register on the next tick of the MSEC system timer register. (See SFRs FDh and FCh)
Delayed load 10	Values are written to the DACxH/DACxL16-bit data buffer, which will be transferred to the DAC output register on the next tick of the HMSEC system timer register. (See SFR FEh)
Synchronized load 11	Values are written to the DACxH/DACxL16-bit data buffer, which will be transferred to the DAC output register when 11_2 is (re)written to these bits.

Direct load mode 00_2 provides a simple means of updating DACs in an arbitrary order and at various times according to the flow of the user's program. For a particular DAC, it is essential that DACH is written before DACL. The code sequence to write C147h to DAC2 in mode 00_2 is shown in Example 7−1.

Example 7−1. DAC Loading

C		Assembler	
	`DACSEL = 0x06;`		`MOV DACSEL,#6`
	`DACL = 0x00;`		`MOV DACL,#0`
	`DACSEL = 0x02;`		`MOV DACSEL,#2`
	`DACH = 0xC1;`		`MOV DACH,#0C1H`
	`DACL = 0x47;`		`MOV DACL,#47H`
or	`DAC = 0xC147;`	not	`MOV DACL,#47H`
			`MOV DACH,#0C1H`

In cases where synchronization is essential, three methods are provided. Delayed modes 01_2 and 10_2 assume that all DACs will be updated at regular intervals, as determined by the milliseconds timer (MSEC) or the hundreds of milliseconds timer (HMSEC), respectively. In either of these modes, the program should ensure that all DAC buffers are reloaded as required before the corresponding timer tick. Note that an interrupt service routine may be associated with MSEC but not directly with HMSEC.

For applications where multiple DACs must be updated synchronously under direct program control, mode 11_2 is provided. Once this mode is established, values written to the data registers are transferred to the output registers when mode 11_2 is rewritten to the control bits.

Given that the settling time of the DACs is approximately 8µs, it is possible for all four DACs to be updated by software within this time scale (using mode 0) and cause them to apparently change together. However, in general, this will only be true for environments without interrupts. Care should be taken when using load mode 00_2 with what appears to be sequential updates of different DACs. In terms of program flow, they may appear adjacent, but interrupt activity will cause them to be separated in time.

7.3 DAC Configuration and Control

Each DAC has a corresponding control register DACxCON (x = 0 to 3), which is used to configure its mode of operation, as summarized in Table 7−4.

Table 7−4. DAC Control Registers

DACSEL SFR x	DAC	7	6	5	4	3	2	1	0	Reset Value
sel = 04h SFR B5h	0	COR0	EOD0	IDAC0DIS	0	0	SELREF0	DOM0_1	DOM0_0	63h
sel = 04h SFR B6h	1	COR1	EOD1	IDAC1DIS	0	0	SELREF1	DOM1_1	DOM1_0	63h
sel = 05h SFR B5h	2	0	0	0	0	0	SELREF2	DOM2_1	DOM2_0	03h
sel = 05h SFR B6h	3	0	0	0	0	0	SELREF3	DOM3_1	DOM3_0	03h

where

Bit	Name	Meaning(s)
COR0 COR1	Current Over Range	Write: 0: Release from high-impedance state back to normal mode unless an over-range (still) exists. Write: 1: NOP Read: 0: IDACx is not over-current Read: 1: IDACx is over 125mA. If EODx = 0, the indication is immediate. If EODx = 1, the over-current condition must occur for three consecutive ticks of MSEC.
EOD0 EOD1	Enable Over-Current Detection	0: Disable over-current detection. 1: Enable over-current detection (default).
SELREFx	Select Reference	0: DACx reference is AV_{DD} (default). 1: DACx reference is REFOUT/REF IN+ pin (see SFR DCh).

and

DOMx[1:0]	Voltage VDACx, x = 0,1,2,3	Current IDACx, x = 0,1
00	Normal output	IDAC controlled by IDACxDIS
01	Output off 1k to AGND	IDAC off
10	Output off 100k to AGND	IDAC off.
11	Output off high impedance (default)	IDAC off (default)

Under normal operating conditions the maximum current output of either IDAC0 or IDAC1 should be no more than 25mA, as set by V_{REF}/R_{REF}, with the additional constraints that V_{REF} is no more than 2.5V and AV_{DD} is at least 1.5V above V_{REF}. CORx will be set when the current reaches approximately 125mA, with a range of 50mA to 225mA due to process variations. If a fault condition is to be triggered by CORx, ensure that the current capability of AV_{DD} supply is sufficiently large.

When EODx is 1 and an over-current condition is detected, CORx will change to 1 and the DAC outputs (current and voltage) will be disabled until released by writing a ‘0’ to CORx.

IDACxDIS and DOMx bits are not altered by the over-current protection mechanism.

If EODx is 0, and depending upon the specific application, the program should poll the CORx bit to confirm that an over-current condition does not exist.

7.4 DAC Technology and Limitations

The DACs in the MSC1211 are based upon the 16-bit DAC type DAC8531, also manufactured by Texas Instruments. Consequently, all DACs use a string of tapped resistors to establish a scale of voltages that are ensured to be monotonic, which is essential for many closed-loop control systems. This is effectively 65,536 resistors that can be tapped for voltages from AGND to the DAC reference voltage.

The output amplifiers for the DACs cannot reach 0V and must have at least 1.5V of operating headroom; that is to say, AV_{DD} should be 1.5V above the maximum voltage output by a DAC. Due to the former constraint, DAC codes below about 0x0200 are not recommended and are precluded from linearity calculations used in the preparation of electrical characteristics, as found in product datasheets. Increased nonlinearity may also be seen for near full-scale codes when the headroom constraint is not satisfied. For example, when a DAC uses the on-chip 2.5V reference and AV_{DD} is less than 4.0V.

The linearity of the DAC can be improved with the techniques discussed in Application Note SBAA112, *MSC1211/12 DAC INL Improvement*, available for download at www.ti.com.

For DACs operating in voltage mode, the reference voltage may extend to AV_{DD} but the output voltage should remain 1.5V lower.

The nominal reference current is 25µA per DAC. Therefore, when the internal voltage reference is disabled and V_{REF} is derived from an external source connected to the REFOUT/REF IN+ pin, the origin and impedance of the source voltage should be considered.

7.5 DAC Example Program

Example 7−2 shows a C program in which a variable, *i*, ranges from 0 to 250 in steps of 10. For each value, 250 × *i*, *i* × *i*, and (40 × *i* / 252) ^ 3 are calculated using only integer arithmetic. The 3 functions are computed at different times, but synchronous load mode 11_2 ensures that DACs 1, 2, and 3 are updated simultaneously; this may be verified with an oscilloscope. Note that the ADC has to be powered to make the internal voltage reference available.

Example 7−2.DAC Program

```
// File DAC04 - Testing DAC on MSC1211 with direct and synchronous loading
// MSC1211 EVM Switches 1:On  SW3-12345678 SW5-12345678
//                      0:Off     11110111     11110000

#include <Reg1211.h>

data unsigned int i,j;

void main(void) {
  PDCON  &= ~0x48;             // Turn on dacs and adc
  ADCON0  =  0x30;             // REFOUT/REFIN+ = Internal 2.5V ref
  DACSEL=6; DACL=0xFC;         // load DACs 3,2,1 simultaneously
  DACSEL=4; DACL=0x24;         // DAC0 IDAC=off, Ref=REFOUT/REFIN+
            DACH=0x24;         // DAC1 IDAC=off, Ref=REFOUT/REFIN+
  DACSEL=5; DACL=0x24;         // DAC2 Ref=REFOUT/REFIN+
            DACH=0x24;         // DAC3 Ref=REFOUT/REFIN+
  while(1) {
    DACSEL=0; DAC=0x8000;      // 1.25 V pulse on DAC0
    for(j=0; j<100; j++);      // delay
    DAC=0;                     // Synchronize 'scope to negative edge
    for(i=0; i<251; i+=10){
      DACSEL=1; DAC=250*i;                   // Linear (DAC1)
      DACSEL=2; DAC=i*i;                     // Square (DAC2)
      DACSEL=3; j=40*i; j=j/252; DAC=j*j*j;  // Cube   (DAC3)
      DACSEL=6; DACL=0xFC;                   // load DACs 3,2,1 simultaneously
    }
  }
}
```

Chapter 8

Pulse-Width Modulator and Tone Generator

This chapter describes the pulse-width modulator (PWM) and tone generator of the MSC121x.

Topic **Page**

8.1 Description 8-2
8.2 PWM Generator Example 8-4

8.1 Description

The PWM subsystem consists of the following components:

- ❑ 6-bit control register PWMCON
- ❑ 16-bit Period register (P) and 16-bit Duty register (D), which share the same SFR addresses
- ❑ 16-bit down-counter, 16-bit comparator, and a finite state machine
- ❑ A single output pin shared with bit 3 of Port 3

The PWM subsystem is enabled by:

1) making bit 4 of PDCON at F1h equal to 0
2) making bit 3 of P3 equal to 1
3) configuring bit 3 of Port 3 to be a standard 8051 port, or open drain. This is achieved by writing 00_2 or 10_2 respectively to bits 7 and 6 of P3DDRL.

The mode of operation is determined by bits within PWMCON, as summarized in Table 8−1.

Table 8−1. PWMCON—PWM Control

PWMCON		**SFR A1h**	**Reset Value = 00h**
Bit #	**Name**	**Action or Interpretation**	
7	−	Not used	
6	−	Not used	
5	PPOL	Period Polarity If 0, the Duty register determines the time the PWM output is high. If 1, the Duty register determines the time the PWM output is low.	
4	PWMSEL	PWM Register Select If 0, data written to PWLHI:PWMLO, at A3h and A2h, respectively, will be directed to the Period register. If 1, data written to PWLHI:PWMLO, at A3h and A2h, respectively, will be directed to the Duty register.	
3	SPDSEL	Speed Select If 1, the down counter is clocked every t_{CLK} seconds. Otherwise, the down counter is clocked every $t_{CLK} \times$ (USEC+1), where USEC is the 5-bit SFR at FBh.	
2−0	TPCNTL	000: Disable	High Impedance = HiZ
		001: PWM	If PPOL is 0, then output is high for D every P and Duty cycle = D/P If PPOL is 1, then output is low for D every P and Duty cycle = (P−D)/P
		011: Square	Low for P; High for P
		111: Staircase	High for (P−Z); HiZ for Z; Low for (P−Z); HiZ for Z

NOTE: P = Period[15:0] + 1; D = Duty[15:0]; Z = Period[15:2] (that is, the integer part of Period divided by 4). For large P, Z is approximately P/4.

The PWM output may be filtered to give a dc level, or used directly in switching systems with inherent filtering to produce a variable effect. Typical examples of the latter are brilliance control of a lamp, power of a heating element, or speed control of a dc motor.

When the staircase mode is used, the output repeats as a (strong 1, High-Z, strong 0, and High-Z).

If a PWM signal is used in a closed-loop, real-time control system, the Duty register will be regularly updated as part of normal operation. Since this 16-bit register is modified by writing to two 8-bit SFRs there is a possibility that either an interrupt will occur between the writes, or the PWM generator will use the Duty register between writes. In either case, one or more cycles may occur with the wrong 16-bit value and cause undesired perturbation of the controlled system. To avoid this possibility, writes to the Duty (or Period, or Tone) registers should be protected from interrupts and/or synchronized with changes on pin P3.3. Since this pin is shared, $\overline{\text{INT1}}$ may be monitored to assist in synchronization in PWM and square-wave modes.

In PWM mode, if the value in the Duty register is larger than that in the Period register, the output is held either low or high depending on the state of PPOL (bit 5) in PWMCON.

Table 8−2. PWM Output

PPOL	Condition	Duty Cycle % High
0	Period = X, Duty = 0	0
0	0 < Duty ≤ Period	Intermediate value
0	Duty > Period	100
1	Period = X, Duty = 0	100
1	0 < Duty ≤ Period	Intermediate value
1	Duty > Period	0

8.2 PWM Generator Example

Example 8−1 shows how the generator can be configured in PWM, square, or staircase modes. It also indicates how to use the system timers to produce an interrupt every second. Once a particular mode is selected after ‘reset’, it should not be changed until after another ‘reset’. However, any mode may be disabled and then restored.

Example 8−1. PWM Generator

```
// File TONE4.c - Testing Tone generator
// MSC1211 EVM Switches 1:On  SW3-12345678 SW5-12345678
//                      0:Off     11110111     11110000
#include <Reg1211.h>
#include <stdio.h>
#define xtal 22118400
#define divA xtal/440    // Concert pitch 'A'
sbit RedLed    = P3^4;   // RED LED on EVM
sbit YellowLed = P3^5;   // Yellow LED on EVM

data unsigned char i,tout;
typedef enum {null, pwm, square, staircase} pwmtype;
code at 0xFFF3 void autobaud(void);

/* Auxiliary Interrupt */
void AuxInt(void) interrupt 6 using 1
{ char temp;
  YellowLed=!YellowLed;
  if (tout) tout--;
  temp=SECINT;           // remove seconds interrupt flag
  EICON&=~0x10;          // remove AI flag
}

void beep(unsigned int divisor, unsigned char time, pwmtype type)
{ PWMCON&=~0x37;         // POL=0, PWMSEL=0, disable
  TONE=divisor;          // Period register
  switch(type) {
    case null: break;
    case pwm:
     { PWMCON|=0x10;     // Duty register
       TONE=9*(unsigned long)divisor/10;
       PWMCON|=1;        // pwm
       break; }
    case square:
```

Example 8−1. (Continued)

```
     { PWMCON|=3;        // square
       break; }
    case staircase:
     { PWMCON|=7;        // staircase
       break; }
    }
  tout=time;
  while(tout);
}

void main(void)
{ PDCON&=~0x12;          // power up PWM generator and seconds
  tout=0;                // time-out is over
  MSEC=xtal/1000-1;      // 1ms tick
  HMSEC=100-1;           // 100ms tick
  SECINT=0x89;           // write 9 immediately for 10 x 100 ms
  RedLed=0;              // indicate start of autobaud
  autobaud();            // set up serial rate
  AIE=0x80;              // enable Seconds interrupt
  EICON|=0x20;           // enable auxiliary interrupt
  PWMCON=0x08;           // PWMSEL=Period Register, fclk
  INT1=1;                // Pin P3.3 is high
  P3DDRL&=~0xC0;         // 8051 output
  RedLed=1;              // indicate waiting for carriage return
  while(1){
    printf("\nPress 1 (PWM), 2 (SQUARE) or 3 (STAIRCASE)\n");  // prompt
    RI_0 = 0;            // wait for character
    while(!RI_0);
    i=SBUF0&3;           // limit range
    RI_0=0;
    printf("Tone in Progress...");
    switch(i) {
      case 0 : break;
      case 1 : {
        beep(divA-1,3,pwm);       // parameters computed at compile-time
        break; }
      case 2 : {
        beep(divA/2-1,4,square);  // divA/2-1 is 25133.54, truncated to 0x622d
        break; }
      case 3 : {
```

Example 8−1. (Continued)

```
        beep(divA/2-1,5,staircase);
        }
      }
    printf("Tone Complete\n");
    beep(0,1,null);
    printf("Press Enter or <cr> \n\n");
    SRST=1; SRST=0;      //RESET
    }
}
```

Chapter 9

Inter-IC (I^2C ™) Subsystem

This chapter describes the Inter-IC (I^2C) subsystem of the MSC1211 and MSC1213.

Topic **Page**

9.1 Introduction to the I^2C Bus 9-2
9.2 I^2C Terminology 9-2
9.3 I^2C Bus Lines and Basic Timing 9-3
9.4 I^2C Data Transfers and the Acknowledge Bit 9-4
9.5 I^2C Principal Registers 9-6
9.6 I^2C Realted Registers 9-10
9.7 I^2C Example—MSC1211/13 as Master 9-11
9.8 I^2C Example—MSC1211/13 as Slave 9-13
9.9 I^2C Example—MSC1211/13 as an Interrupt-Driven Slave 9-15
9.10 I^2C Synchronization and Arbitration 9-17
9.11 I^2C Fast Mode 9-17
9.12 I^2C General Call 9-17
9.13 I^2C 10-Bit Addressing 9-18

9.1 Introduction to the I2C Bus

The MSC1211/13 provide hardware support for serial transfers according to the I2C protocol. This protocol was defined to permit multiple 8-bit transfers between multiple integrated circuits on the same 2-wire bus. At any one time a bus master coordinates transfers from one slave or to multiple slaves.

For a detailed description of the I2C bus, refer to *The I2C-bus Specification* by Philips.

9.2 I2C Terminology

For many systems where the MSC1211/13 is the only microcontroller, it will be the master, and coordinate the transfer of data between itself and slave ICs. If active, it can transmit data onto the I2C bus or receive data from the bus. In either case, it generates the synchronizing clock.

Similarly, where the MSC1211/13 is considered a slave to another microcontroller, it is able to transmit and receive data synchronized by this master.

Many MSC1211/13s can share a single I2C bus where each acts as a master at different times. The active master can be determined by software or result from bus arbitration in the event of asynchronous contention.

Table 9−1 describes selected I2C terms.

Table 9−1. I2C Terminology

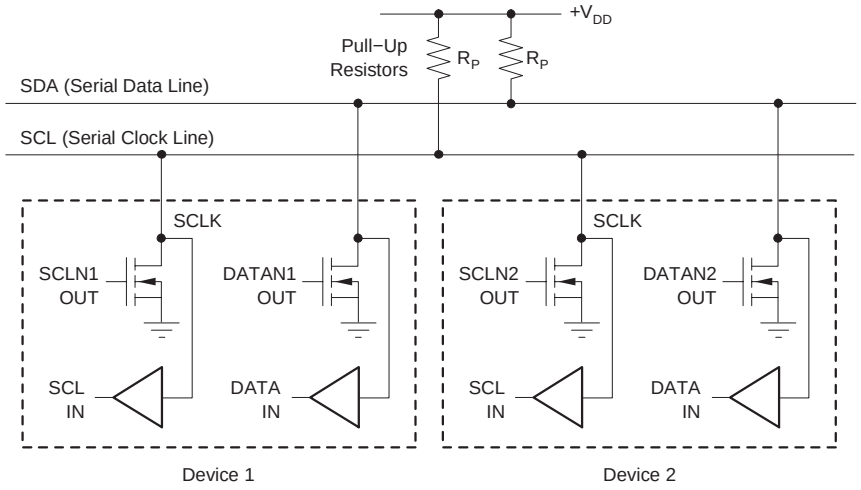
Name	Description
Transmitter	The IC that sends data to the bus
Receiver	The IC that receives data from the bus
Master	The IC that initiates a transfer, generates clock signals, and terminates a transfer
Slave	The IC addressed by a master
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message
Arbitration	Procedure to ensure that if more than one master simultaneously tries to control the bus, only one is allowed to do so and the message is not corrupted
Synchronization	Procedure to synchronize the clock signals of two or more ICs

9.3 I2C Bus Lines and Basic Timing

The I2C bus uses two bidirectional data lines. One is the data line (SDA), and the other is the clock line (SCL). Each is connected to a positive supply voltage via a pull-up resistor, and when the bus is free, both lines are high.

The output stages of I2C interfaces connected to the bus must have an open drain or open collector to perform the wired-AND function. The original specification for the I2C bus allowed the data transfer rate to be up to 100kbits/s; however, this has been extended to 400kbits/s in fast mode, which is supported by the MSC1211/13. In either mode, the maximum rate is determined by the value of the pull-up resistors and the capacitance to ground.

Figure 9−1. I2C Bus Connection of Standard and Fast Mode Devices



Unique START and STOP conditions are identified when SCL is high and SDA changes. If SDA changes from 1 to 0, a START condition is created; if SDA changes from 0 to 1, a STOP condition is created. All ICs connected to the bus, including the MSC1211/13, recognize and respond to START and STOP conditions. For a data-bit transfer, SCL is pulsed high while SDA is stable.

Figure 9−2. START and STOP Conditions

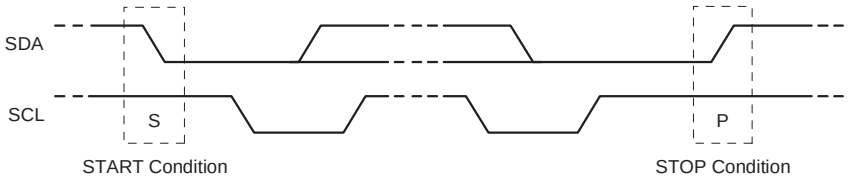
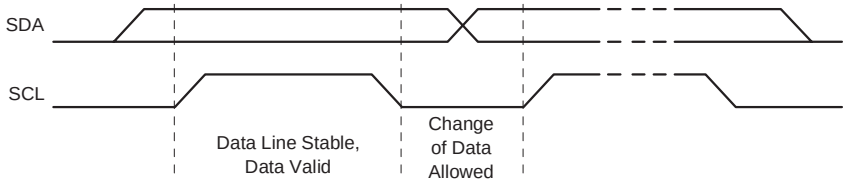


Figure 9−3. I2C-Bus Bit Transfer



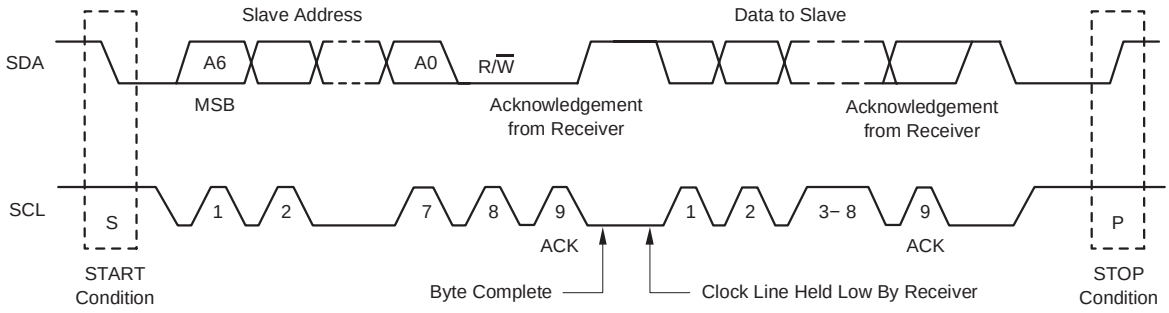
9.4 I2C Data Transfers and the Acknowledge Bit

Once a master asserts a START condition, the bus is no longer free. The master then transmits eight bits comprising of the 7-bit address of the slave followed by a read/write (R/$\overline{W}$) bit. In a system with multiple asynchronous masters, there may be a period of bus contention and arbitration before the address of the slave is transmitted.

If the slave is to receive data, the R/$\overline{W}$ bit must be 0; otherwise, it will prepare to transmit data (since the R/$\overline{W}$ bit is 1). For some I2C devices, such as memories, it is necessary to first write an internal address to the slave and then read or write data bytes. In this case, a START condition can be re-asserted.

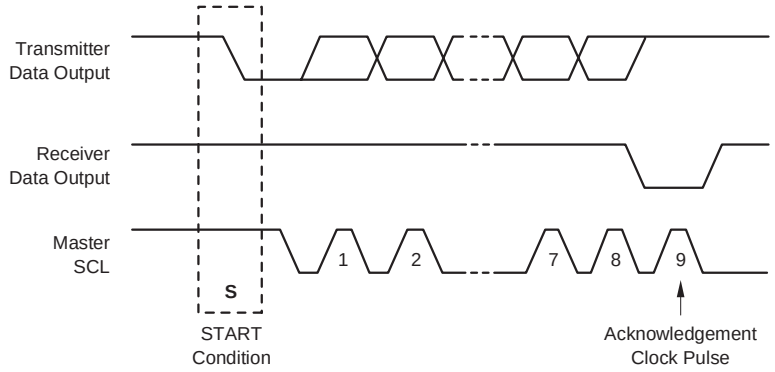
When a master has generated eight SCL pulses, it places its own SDA output high and generates a ninth clock pulse. If the addressed slave has responded, it will have pulled the SDA line low; this represents an acknowledgement (ACK). However, if the addressed slave leaves the SDA line high, the master recognizes that the slave has not acknowledged (NACK) the transfer.

Figure 9–4. I2C-Bus Data Transfer



Once addressed, a multi-byte data transfer can be terminated when a slave generates a NACK rather than the usual ACK. In addition, after the acknowledge bit, a slave may pull the SCL line low while it performs local processing; this often occurs when the slave is a microcontroller that executes a time-consuming interrupt service routine. While the SCL line is held low, the master will wait.

Figure 9–5. I²C Acknowledge



If a master issues a slave address with a R/$\overline{W}$ bit that is 1, it will become a master receiver when the slave responds with an ACK. Thereafter, the slave provides data bytes to the master, but releases the SDA line every ninth clock pulse and samples the acknowledgement that is provided by the master. Typically, the master will generate ACKs for as long as it expects more data, and then generate a NACK to inform the slave on the last byte.

9.5 I2C Principal Registers

There are four principal Special Function Registers (SFRs) associated with the MSC1211/13 I2C interface. These SFRs are:

I2CCON at 9Ah—provides primary control

I2CDATA at 9Bh—provides data

I2CGCM at 9Ch—provides additional control

I2CSTAT at 9Dh—provides status

The I2C Control register (I2CCON) is described in Table 9−2.

All I2C bytes are written to, or read from I2CDATA. When the byte written represents an I2C slave address between 0001000_2 and 1111011_2, bit 0 is the R/$\overline{W}$ flag, such that R/$\overline{W}$ = 1 for read and R/$\overline{W}$ = 0 for write (see Table 9−3).

Bit 7 of I2CGM at 9Ch is used to control the behavior of a slave to the General Call address, or to allow multiple masters (see Table 9−4).

Table 9−2. I2CCON—I2C Control Register

I2CCON		**SFR 9Ah** **Reset Value = 00h**
Bit #	**Name**	**Action or Interpretation**
7	START valid only if MSTR = 1	Read: Current status of START condition or repeated START condition Write: 0: No action. 1: Transmit a START condition. If the bus is not free, a START condition will be transmitted after a STOP condition has been received. If the START bit is set after at least one byte has been transmitted, a repeated START condition is transmitted after the current data transfer is completed. If both START and STOP are set while the bus is free, a START condition will be followed by a STOP condition.
6	STOP valid only if MSTR = 1	Read: Current status of STOP condition. Write: 0: No action 1: Transmit a STOP condition. If both START and STOP are set during a data transfer, a STOP condition is transmitted followed by a START condition.
5	ACK	Defines the type of acknowledgement generated during the acknowledge cycle. Master or slave receiver write: 0: Not acknowledge (NACK, high level on SDA) is generated. 1: Acknowledge (ACK, low level on SDA) is generated. Slave transmitter write: 0: The current byte will be the last byte transmitted because NACK occurs. 1: One or more bytes to follow the current transaction because ACK occurs.
4	0	Reserved. Always write 0.
3	FAST	Write: 0: Standard Mode (100 kHz). 1: Fast Mode (400 kHz).

Table 9−2. I2CCON—I2C Control Register (continued)

I2CCON		SFR 9Ah — Reset Value = 00h
Bit #	**Name**	**Action or Interpretation**
2	MSTR	Write: 0: Slave Mode. 1: Master Mode.
1	SCLS	Write: 0: No effect 1: Remove stretch of SCL low, when in slave mode. Allowed only after I2C master has put SCL low.
0	FILEN	50ns glitch filter. Write: 0: Filter disabled. 1: Filter of approximately 50ns is enabled.

Table 9−3. I2CDATA SFR

I2CDATA SFR 9Bh	7	6	5	4	3	2	1	0	Reset Value
Data	MSB							LSB	00h
Address	MSB						LSB	R/W	00h

Table 9−4. I2CGM—I2C General Call / Multiple Master Control

I2CGM		SFR 9Ch — Reset Value = 00h
Bit #	**Name**	**Action or Interpretation**
7	GCMEM	Write only. Slave mode write: 0: General Call address will be ignored 1: General Call address will be detected. Master mode write: 0: Single Master 1: Multiple Master mode.

In master mode, a write to I2CSTAT sets the frequency of the SCL line to SYSCLK/[2 × (SCKD + 1)], where the minimum value allowed for SCKD is 3. In slave mode, a write to I2CSTAT sets the slave address in bits 6 to 0, which is only recognized if Slave Address Enable (SAE) is 1. Table 9−5 shows the I2CSTAT SFR.

In either mode, reading I2CSTAT returns a 5-bit status code that is left-justified and clears the I2C Status Interrupt flag in bit 2 of AIE. Left-justified status codes can be used to implement jump table easily. The status codes are listed in Table 9−6.

Table 9−5. I2CSTAT SFR

I2CSTAT SFR 9Dh	7	6	5	4	3	2	1	0	Reset Value
Read	STAT7	STAT6	STAT5	STAT4	STAT3	0	0	0	x
Write	SAE	SA6	SA5	SA4	SA3	SA2	SA1	SA0	00h
Write	SCKD7	SCKD6	SCKD5	SCKD4	SCKD3	SCKD2	SCKD1	SCKD0	x

Table 9−6. I2C Status Codes

State	Status Code (Hex)	Mode	Action Taken by the I2C
00	00	Waiting	No action.
01	08	Master Transmitter/Receiver	START condition transmitted.
02	10	Master Transmitter/Receiver	Repeated START condition transmitted.
03	18	Master Transmitter	Slave address + W(1) transmitted. ACK received.
04	20	Master Transmitter	Slave address + W transmitted. NACK received.
05	28	Master Transmitter	Data byte transmitted. ACK received.
06	30	Master Transmitter	Data byte transmitted. NACK received.
07	38	Master Transmitter	Arbitration lost.
08	40	Master Receiver	Slave address + R(1) transmitted. ACK received.
09	48	Master Receiver	Slave address + R transmitted. NACK received
0A	50	Master Receiver	Data byte received. ACK transmitted.
0B	58	Master Receiver	Data byte received. NACK transmitted.

1) +W means R/$\overline{W}$ bit is 0; +R means that R/$\overline{W}$ bit is 1.

Table 9−6. I2C Status Codes (continued)

State	Status Code (Hex)	Mode	Action Taken by the I2C
0C	60	Slave Receiver	Own slave address + W already received. ACK transmitted.
0D	68	Invalid	Reserved
0E	70	Slave Receiver	General call address (00h) received. ACK returned.
0F	78	Invalid	Reserved
10	80	Slave Receiver	Own slave address + W already received. Data byte received. ACK transmitted.
11	88	Slave Receiver	Own slave address + W already received. Data byte received. NACK transmitted.
12	90	Slave Receiver	General call address (00h) received. Data byte received. ACK transmitted.
13	98	Slave Receiver	General call address (00h) received. Data byte received. NACK transmitted.
14	A0	Slave Receiver	A STOP or repeated START received while addressed as a slave or General Call
15	A8	Slave Transmitter	Own slave address + R already received. ACK transmitted.
16	B0	Invalid	Reserved
17	B8	Slave Transmitter	Data byte transmitted. ACK received.
18	C0	Slave Transmitter	Data byte transmitted. NACK received.
19	C8	Slave Transmitter	Last data byte transmitted; will switch to non-addressed slave.
1A to 1E	D0−F0	Invalid	Reserved
1F	F8	Invalid	Reserved

1) +W means R/$\overline{W}$ bit is 0; +R means that R/$\overline{W}$ bit is 1.

9.6 I2C Related Registers

The I2C interface shares pins and registers with the Serial Peripheral Interface (SPI); both interfaces must not be enabled at the same time via bit 5 (PDI2C) and bit 0 (PDSPI) of Power-Down Control (PDCON) SFR at F1h.

Table 9−7. PDCON of I2C and SPI

PDCON at F1h			
Bit 5 = PDI2C	**Bit 0 = PDSPI**	**I2C**	**SPI**
0	0	Undefined	Undefined
0	1	Enabled	Disabled
1	0	Disabled	Enabled
1	1	Disabled	Disabled

The I2C interface uses bit 2 (EI2C) of the Auxiliary Interrupt Enable (AIE) SFR at A6h to enable interrupts, as well as bit 2 (I2CSI) of the Auxiliary Interrupt Status Register (AISTAT) SFR at A7h and bit 4 (AI) of the Enable Interrupt Control (EICON) SFR at D8h.

The setup and hold times for data transfers are determined by the frequency (*f*) of the MSC1211/13 oscillator and the value written to the USEC SFR at FBh. It is expected that USEC is set to (*f* − 1), where *f* is in MHz, so that an internal reference of approximately 1µs is obtained.

Table 9−8. Interrupt Control for I2C

SFR Name	SFR Address	Bit Number	Bit Name	Action or Interpretation
AIPOL	A4h	2	I2C	I2C Status Interrupt (before masking) Read: 0: I2C interrupt inactive 1: I2C interrupt active
PAI	A5h	3,2,1,0		Pending Auxiliary Interrupt Register Read 0011_2 indicates I2C interrupt pending
AIE	A6h	2	EI2C	Enable I2C Status Interrupt. Write: 0: Masked 1: Enabled (shared vector to address 0033h) Read: Current value of I2C status interrupt before masking.
AISTAT	A7h	2	I2CSI	I2C Status Interrupt Read: 0: I2CSI interrupt inactive or masked 1: I2CSI interrupt active
EICON	D8h	5	EAI	Enable Auxiliary Interrupt. The Auxiliary Interrupt accesses nine different interrupts that are masked by AIE (SFR A6h) and identified by AISTAT (SFR A7h) and PAI (SFR A5h). Write: 0: Auxiliary Interrupt disabled (default) 1: Auxiliary Interrupt enabled.
EICON	D8h	4	AI	Auxiliary Interrupt Flag When PAI indicates that there are no pending auxiliary interrupts (that is, all auxiliary interrupts have been serviced), AI must be cleared by software before exiting the interrupt service routine; otherwise, the interrupt will occur again. Setting AI in software generates an auxiliary interrupt, if enabled. 0: No Auxiliary Interrupt detected (default) 1: Auxiliary Interrupt detected.

9.7 I2C Example—MSC1211/13 as a Master

In order to transmit or receive data via the I2C bus, the programmer must write code that generates the sequence of transfers required by each particular I2C device. The transition between states is reflected in the I2C status codes returned via I2CSTAT. Depending upon the frequencies of the system clock and SCL as well as overall complexity, the programmer may choose to use inline code or make use of interrupt structures. Care should be taken to account for all possible state transitions, in case the program becomes stuck waiting for a condition that does not occur; for example, when an unexpected NACK is received.

The program uses the MSC1211/13 to coordinate data transfers between a real-time clock (PCF8593) and an I/O port (PCF8574A) to cause its bit 7 to pulse once per second. The control byte at address zero within the PCF8593 is repeatedly redefined and while this is not strictly necessary, it is convenient in Example 9−1. After writing this byte, the internal address is automatically incremented so that it points to the fractions-of-a-second register.

Example 9−1.MSC1211 as a Master

```
// Program RTCIO_02.c
// MSC1211 to/from Philips PCF8593 Real Time Clock at address A2 and
// PCF8574A 8 bit I/O port at address 7E
// Including synchronisation with SCL
#include ”stdio.h”
#include ”REG1211.h”
#PRAGMA NOIP

code at 0xFFF3 void autobaud(void);
char i,i1,i2,i3;             // global Variables

void main() {

   PDCON   = 0x5F;           // enable I2C alone
   autobaud(); printf(”I2C RTC to IO \n\n”);
   RI_0    = 0;
   USEC    = 21;             // divide by 22
   I2CCON  = 0x04;           // NACK, 0, Normal,
                             // Master, No stretch, Not Filtered
   I2CGM   = 0x00;           // single master
   I2CSTAT = 0x6D;           // for 22MHz osc, 100 kHz clock

   while (1){
     while (!RI_0) {                 // continue until serial character
       I2CCON|= 0x80;                // START
       while (!(AIE&0x04));          // wait for I2C interrupt flag
       i=I2CSTAT; if(i!=0x08) break; // handle unexpected condition
       while (SCL);                  // wait

       I2CDATA = 0xA2;               // Slave address with write bit
       while (!(AIE&0x04));          // wait for I2C interrupt flag
       i=I2CSTAT; if(i!=0x18)break;  // handle unexpected condition

       I2CDATA=0x00;                 // Address within PCF8593
       while (!(AIE&0x04));          // wait for I2C interrupt flag
       i=I2CSTAT; if(i!=0x28) break; // handle unexpected condition
```

Example 9−1. (Continued)

```
      I2CDATA=0x00;                  // Control byte => 32768 osc
      while (!(AIE&0x04));           // wait for I2C interrupt flag
      i=I2CSTAT; if(i!=0x28) break;  // handle unexpected condition

      I2CCON|= 0x40;                 // STOP request
      while(i=I2CCON,(i&0x40));      // wait for stop to occur

      I2CCON|= 0x80;                 // START request
      while (!(AIE&0x04));           // wait for I2C interrupt flag
      i=I2CSTAT; if(i!=0x08) break;  // handle unexpected condition
      while (SCL);                   // wait

      I2CDATA = 0xA3;                // slave address with read bit
      while (!(AIE&0x04));           // wait for I2C interrupt flag
      i=I2CSTAT; if(i!=0x40) break;  // handle unexpected condition

      I2CCON|=0x20;                  // with ACK
      i=I2CDATA;                     // read byte to trigger data transfer
      while (!(AIE&0x04));           // wait for I2C interrupt flag
      i=I2CSTAT; if(i!=0x50) break;  // handle unexpected condition
      i1=I2CDATA;                    // read 'fractions of seconds'

      while (!(AIE&0x04));           // wait for I2C interrupt flag
      i=I2CSTAT; if(i!=0x50) break;  // handle unexpected condition

      I2CCON&=~0x20;                 // with NACK
      i2=I2CDATA;                    // read 'seconds'
      while (!(AIE&0x04));           // wait for I2C interrupt flag
      i=I2CSTAT; if(i!=0x58) break;  // handle unexpected condition

      i3=I2CDATA;                    // read 'minutes'

      I2CCON|=0x40;                  // STOP request
      while(i=I2CCON,(i&0x40));      // wait for stop to occur

      I2CCON|= 0x80;                 // START
      while (!(AIE&0x04));           // wait for I2C interrupt flag
      i=I2CSTAT; if(i!=0x08) break;  // handle unexpected condition
      while(SCL);                    // wait

      I2CDATA = 0x7E;                // slave address with write bit
      while (!(AIE&0x04));           // wait for I2C interrupt flag
      i=I2CSTAT; if(i!=0x18) break;  // handle unexpected condition

      I2CDATA=~(i1&0x80);            // value for P8547
      while (!(AIE&0x04));           // wait for I2C interrupt flag
      i=I2CSTAT; if(i!=0x28) break;  // handle unexpected condition

      I2CCON|= 0x40;                 // STOP request
      while(i=I2CCON,(i&0x40));      // wait for stop to occur
      i=0xFF;                        // flag valid termination;
    }
   if(i!=0xFF)
    { printf("unexpected condition %4d to be handled \n",i); break;}
   RI_0 = 0;
   while (!RI_0); RI_0 = 0;          // wait for character
  }
  while(1);                          // endless loop
}
```

9.8 I2C Example—MSC1211/13 as a Slave

When operating as a slave, data may be received, transmitted, or both. In Example 9−2, two bytes are received from a master, and their AND and OR are sent back. To simulate the time taken for additional computations encountered in most real applications, a delay is introduced to emphasize the effect of a *stretched* clock, when the slave holds SCL low. The I2C Status Interrupt flag in AIE is set as a result of the positive edge of SCL during the Acknowledge bit, but SCL is not stretched until the negative edge. The SCLS bit in I2CCON must be set in order to release the SCL line, but this must not occur until the clock is being stretched.

The C code uses a switch statement to create a multi-way branch for each of the expected status codes. More efficient code may be created using assembler language.

Example 9−2.MSC1211/13 as a Slave

```
// Slave04.c
// I2C master to/from slave MSC1211 at address 1110100
// Returned data are functions of received data.
#include ”stdio.h”
#include ”REG1211.h”
#PRAGMA NOIP

code at 0xFFF3 void autobaud(void);

char i,r1=0,r2=0,t1=1,t2=2; //global Variables
int j;

void delay(void) {for(j=0;j<1000;j++);}

void release(void)  {
while(SCL);                 // ensure clock is low
I2CCON|=0x02; }             // set clock stretch release bit

void process_data(void) {
t1=r1 & r2;                 // AND
t2=r1 | r2;                 // OR
delay();   }                // simulate additional processing time

void main() {
   PDCON   = 0x5F;          // enable I2C alone
   autobaud(); printf(”MSC1211 as an I2C slave \n\n”);
   RI_0    = 0;

   USEC    = 21;            // divide by 22
   I2CCON  = 0x20;          // ACK, 0, Normal, Slave, No stretch, Not Filtered
   I2CGM   = 0x00;          // General Call Ignored
   I2CSTAT = 0xF4;          // Master ’sees’ slave at E8
```

Example 9−2. (Continued)

```
   while (1){
     while (!RI_0) {        // continue until serial character
       I2CCON|= 0x20;       // ACK
       while (!(AIE&0x04)); // wait for I2C interrupt flag
       i=I2CSTAT;           // get status and clear I2C interrupt flag
       switch(i) {
       /* slave address+W */ case 0x60 : release(); break;
       /* received data   */ case 0x80 : r1=r2; r2=I2CDATA; release(); break;
       /* STOP */            case 0xA0 : break;
       /* slave address+R */
         case 0xA8 : process_data(); I2CDATA=t1; release(); break;
       /* transmit data + ACK  */
         case 0xB8 : I2CCON&=~0x20; I2CDATA=t2; release(); break;
       /* transmit data + NACK */
         case 0xC0 : release(); break;
       default   :
         printf("Unexpected condition %4d to be handled \n",i); while(1);
       }
     }
   RI_0 = 0;
   while (!RI_0); RI_0 = 0; // wait for character
  }
}
```

9.9 I2C Example—MSC1211/13 as an Interrupt-Driven Slave

In many applications, I2C communications occur via interrupts, as shown in Example 9−3. It provides the same functional behavior as Example 9−2, except the MSC1211/13 is free to run a foreground task.

Example 9−3.MSC1211/13 as an Interrupt-Driven Slave

```
// Slave04i01.c – Using interrupts
// I2C master to/from slave MSC1211 at address 1110100
// returned data are functions of received data. Common 'release' mechanism
#include "stdio.h"
#include "REG1211.h"
#PRAGMA NOIP

code at 0xFFF3 void autobaud(void);

char r1=0,r2=0,t1=1,t2=2;   // global Variables
int j;

void delay(void) {for(j=0;j<1000;j++);}

void release(void) {
while(SCL);                 // ensure clock is low
I2CCON|=0x02; }             // set clock stretch release bit

void process_data(void) {
t1=r1 & r2;                 // AND
t2=r1 | r2;                 // OR
delay();  }                 // simulate additional processing time

void Aux_Int(void) interrupt 6 using 1 {
char i;
  i=PAI;                    // Auxiliary Interrupt status code
  if(i==3){
    I2CCON|= 0x20;          // ACK
    i=I2CSTAT;              // get status and clear I2C interrupt flag
    switch(i) {
      /* slave address+W */ case 0x60 : release(); break;
      /* received data   */ case 0x80 : r1=r2; r2=I2CDATA; release(); break;
      /* stop */            case 0xA0 : break;
      /* slave address+R */
           case 0xA8 : process_data(); I2CDATA=t1; release(); break;
      /* transmit data + ACK  */
           case 0xB8 : I2CCON&=~0x20; I2CDATA=t2; release(); break;
      /* transmit data + NACK */
           case 0xC0 : release(); break;
           default   :
```

Example 9−3. (Continued)

```
            printf("Unexpected condition %4d to be handled \n",i); while(1);
      }
    AI=0;                   // clear Auxiliary Interrupt flag }
  } else
    {printf("Unexpected interrupt %4d to be handled \n",i); while(1);}
}

void main() {
  PDCON   = 0x5F;           // enable I2C alone
  autobaud(); printf("MSC1211 as an I2C slave using interrupts \n\n");
  RI_0    = 0;
  USEC    = 21;             // 22MHz xtal, Divide by 22 to give 1 us
  I2CCON  = 0x20;           // ACK, 0, Normal, Slave, No stretch, Not Filtered
  I2CGM   = 0x00;           // General Call Ignored
  I2CSTAT = 0xF4;           // Master 'sees' slave at E8

  AIE     = 0x04;           // I2C Status Interrupt Enable
  AI      = 0;              // ensure auxiliary interrupt flag is clear
  EAI     = 1;              // enable auxiliary interrupts

  while (1){
    while (!RI_0) {         // continue until serial character
      putchar('a');         // a foreground task !
      }
    putchar(SBUF);
    RI_0 = 0;
    while(!RI_0); RI_0 = 0; // wait for character
    }
}
```

9.10 I2C Synchronization and Arbitration

Byte-level synchronization is achieved when an MSC1211/13, acting as a slave, holds SCL low after the ninth bit of any byte transferred. However, bit-level synchronization is also supported when an MSC1211/13 is configured as a master, and a slave pulls SCL low after each bit. In effect, it will pause if it senses a low level on SCL when it should be high. More generally, the SCL clock has a low period determined by the device with the longest clock low period, and a high period determined by the device with the shortest high period.

In a system with multiple masters, there is a chance that two or more masters will attempt to place data on SDA at the same time. When one master attempts to transmit a high level while another is already transmitting a low level, it will disable its output stage and relinquish control of SDA. The I2C Status Code of the losing master shows loss of arbitration, while the winning master is left to control the bus and pass error-free data.

9.11 I2C Fast Mode

Assuming USEC is defined to give an internal reference of 1µs, and bit 3 of I2CCON is clear, the I2C subsystem will generate standard setup and hold times. However, if bit 3 of I2CCON is set, this timing will be altered to permit transfers at up to 400kHz, as determined by the value written to I2CSTAT. In fast mode, the SCL and SDA inputs incorporate Schmitt triggers and spike suppression, as well as active slew-rate control of falling edges. Compared with standard mode, it may be necessary to reduce the value of pull-up resistors and/or load capacitance. In exceptional cases, the pull-up may be a high-speed active current source of up to 3mA. For bus loads up to 400pF, the pull-up resistor can be a current source of up to 3mA or a switched resistor circuit.

9.12 I2C General Call

When bit 2 of I2CCON is 0, the MSC1211/13 is configured as a slave device. In this state, if bit 7 of I2CGM at 9Ch is 1 or 0, a general call address of 00h will be recognized or ignored, respectively. When recognized, the status code is set to 70h, and the slave should respond to the following data byte according to the I2C standard.

9.13 I2C 10-Bit Addressing

The original 7-bit addressing scheme of the I2C standard allocates addresses according to Table 9−9.

Table 9−9. Address Allocation

Most Significant Seven Bits	R/$\overline{W}$ Bit	Standard Meaning	Extended Meaning, Where Different
0000 000	0	General call	
0000 000	1	Start byte for slow devices	
0000 001	x	Address for CBUS protocol	
0000 010	?	Address reserved for different protocol	
0000 011 to 0000 111	x	To be defined	
0001 000 to 1110 111	R/$\overline{W}$	I2C device addresses	
1111 0aa	R/$\overline{W}$	Reserved	Most significant two bits of 10-bit address
1111 1xx	x	Reserved	

To increase the number of addressable devices, the I2C standard was extended to accommodate an additional 10-bit address space. The most significant two bits are contained within addresses that were originally reserved, while the remaining eight bits are provided in the following byte. The MSC1211/13 neither generates nor accepts 10-bit addresses automatically. However, the 10-bit addressing protocol may be replicated under software control to implement either a master transmitter or a slave receiver. A master receiver or slave transmitter cannot be implemented because the interpretation of the R/$\overline{W}$ flag precludes transmission or reception (respectively) of the low part of the address as a data byte.

Chapter 10

Serial Peripheral Interface (SPI™)

This chapter describes the serial peripheral interface (SPI) of the MSC121x.

Topic **Page**

10.1 Description 10-2
10.2 SPI Configuration 10-2
10.3 SPI Interrupts 10-5
10.4 SPI FIFO Buffer 10-6
10.5 SPI Examples 10-9

10.1 Description

The Serial Peripheral Interface, or SPI, is a synchronous bit, serial, full-duplex communications standard that simultaneously transfers eight bits of data from a master to a slave, and another eight bits of data from the slave to the master. The MSC121x can be programmed to behave as a master or a slave and uses four signals to coordinate transfers. These signals are:

1) $\overline{SS}$—Slave Select (shared with P1.4/INT2)

2) MOSI—Master Out/Slave In (shared with P1.5/$\overline{INT3}$)

3) MISO—Master In/Slave Out(shared with P1.6/INT4/SDA)

4) SCK—SPI Clock (shared with P1.7/$\overline{INT5}$/SCL)

10.2 SPI Configuration

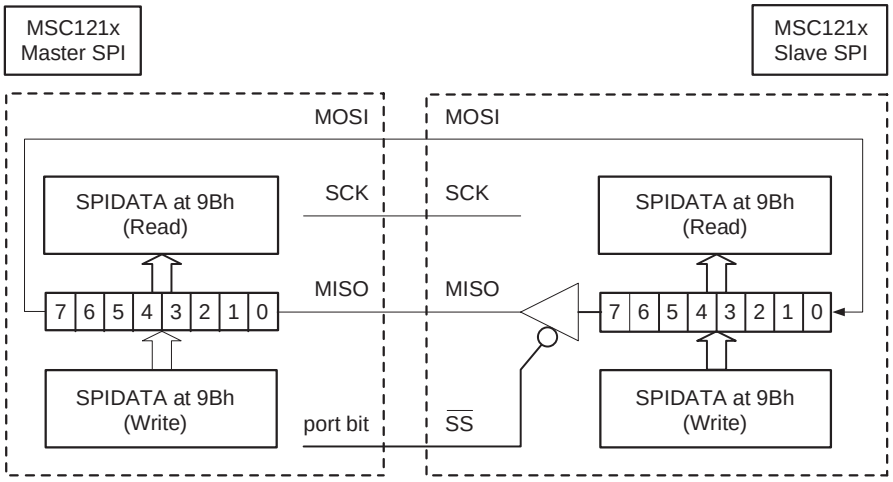
The usual interconnection between a master and slave is shown in Figure 10−1. To multiplex data from more than one slave, the MISO output may be selectively enabled via the active-low slave-select pin. Although less common, the MSC121x permits multiple masters by enabling their MOSI and SCK outputs under software control.

The transmit and receive data pathways are double-buffered, but may also include a first-in/first-out (FIFO) buffer that uses part of the core SRAM. This configuration permits higher-speed transfers and reduces CPU overhead.

To provide compatibility with other slave devices, such as hardware shift registers, the default order of bits transferred can be changed from 7...0 to 0...7. Also, the phase and polarity of the clock (SCK) can be configured.

The SPI subsystem is only active if PDSPI (bit 0) of PDCON at F1h is 0. However, the SPI and I^2C interface (if present) cannot both be enabled because the same SFR addresses are used to support them, with different interpretations according to which interface is powered up.

Figure 10−1. SPI Master/Slave Interconnect



The SPI is configured by SPICON at 9Ah, according to Table 10−1.

Table 10−1. SPICON—SPI Control

SPICON		SFR 9Ah	Reset Value = 00h
Bit #	**Name**	**Action or Interpretation**	
7	SCK2	SCK Selection. SCK = SCK2:SCK1:SCK0 = 000_2 to 111_2	
6	SCK1	SPI clock frequency = $f_{CLK} / 2^{(SCK+1)}$. That is, f_{CLK} / 2 to f_{CLK} / 256 in powers of 2	
5	SCK0	SPI clock period = $t_{CLK} \times 2^{(SCK+1)}$. That is, $t_{CLK} \times 2$ to $t_{CLK} \times 256$ in powers of 2	
4	FIFO	Enable FIFO buffer in core SRAM. 0: Transmit and receive pathways are double buffered 1: Circular FIFO buffer is used for to transmit and receive bytes	
3	ORDER	Sets bit order for transmit and receive. 0: Most significant bit first 1: Least significant bit first	
2	MSTR	SPI Master Mode. 0: Slave mode 1: Master mode	
1	CPHA	Serial clock phase control. 0: Valid data starting from half SCK period before the first edge of SCK 1: Valid data starting from the first edge of SCK	
0	CPOL	CPOL serial clock polarity. 0: SCK idle at logic LOW 1: SCK idle at logic HIGH	

Bits in Port 1 at 90h that are shared with SPI signals should be left in their default states or possibly configured as inputs or CMOS outputs, depending on the signal and mode of operation.

Table 10−2. P1—Port 1

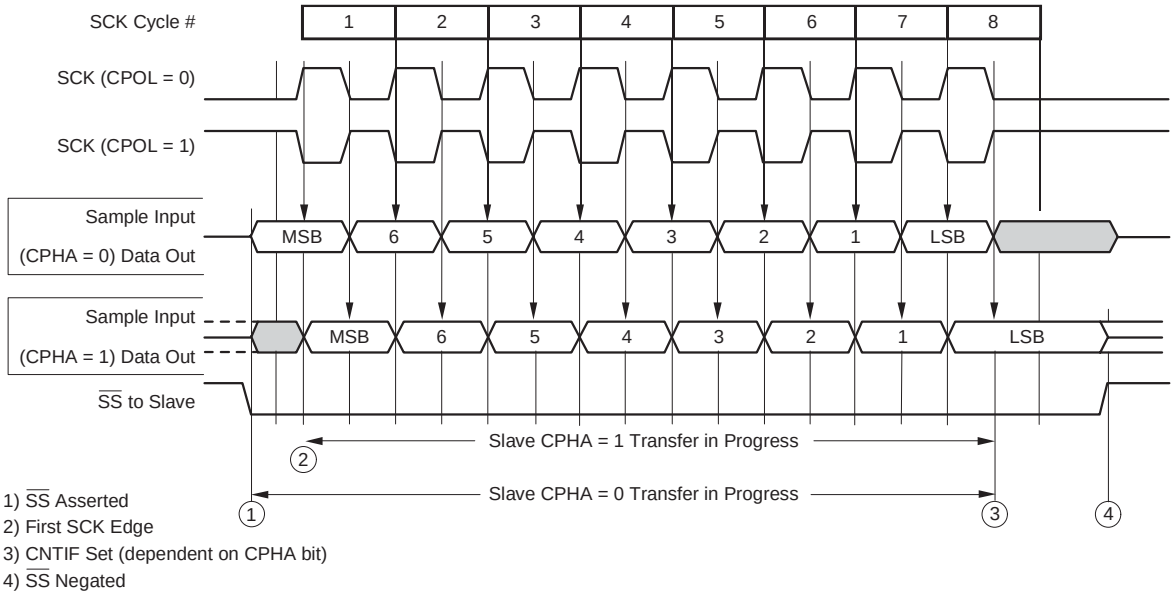
P1		SFR AFh			Reset Value = FFh
		Master		Slave	
Bit #	**Signal**	**Bit Type**	**Port Value**	**Bit Type**	**Port Value**
4	$\overline{SS}$	8051 or CMOS	0 or 1	8051 or input	1
5	MOSI	8051 or CMOS or Open Drain	1	8051 or input	1
6	MISO	8051 or Input	1	8051 or CMOS or Open Drain	1
7	SCK	8051 or CMOS or Open Drain	1	8051 or input	1

NOTE: Bits configured as open drain may require a pull-up resistor.

Table 10−3. P1DDRH—Port 1 Data Direction Register

P1DDRH		SFR AFh	Reset Value = 00h
Bit #	**Name**	**Action or Interpretation**	
7−6	P17H:P17L	Port bit type: 00: Standard 8051 01: CMOS output 10: Open drain output 11: Input	
5−4	P16H:P16L		
3−2	P15H:P15L		
1−0	P14H:P15L		

Figure 10−2. SPI Clock/Data Timing



CPHA and CPOL alter the phase of the data and the polarity of the clock to suit various applications.

Data to be transmitted are written to the SPI Data Register (SPIDATA at 9Bh), which is then passed to the double-buffered SPI transmit interface. Similarly, data that has been received are read via this SFR from the double-buffered SPI receive interface. Data are routed through a FIF0 buffer of up to 128 bytes if bit 4 of SPICON at 9Ah is set.

Table 10−4. SPIDATA—SPI Data Register

SPIDATA		SFR 9Bh	Reset Value = 00h
Bit #	**Name**	**Action or Interpretation**	
7−0	SPIDATA	Write/Read: Data to be transmitted by, or received from, the Serial Peripheral Interface.	

10.3 SPI Interrupts

When an SPI interrupt is active and enabled, the MSC121x CPU jumps to location 0033h. The interrupt service routine may read the Pending Auxiliary Interrupt Register (PAI at A5h) to establish the source of the interrupt. For the SPI receiver, the number returned is 3, and for the SPI transmitter, it is 4. This assumes that higher priority auxiliary interrupts have not occurred.

AI (bit 4 of EICON), must be cleared within the interrupt service routine when no further auxiliary interrupts are pending. Setting AI in software generates an Auxiliary Interrupt, if enabled, but if there are no pending interrupts the Pending Auxiliary Interrupt vector in PIA at A5h will read as 0.

When the FIFO buffer is disabled, the transmit interrupt flag will be set whenever the SPI transmitter is empty and the receiver interrupt flag will be set whenever there is a received byte to read. Writing to SPIDATA will clear the transmit interrupt flag, if previously set. Similarly, reading SPIDATA will clear the receive interrupt flag, if previously set. Because of the bit-synchronous nature of the SPI, a byte is only received when one is transmitted. Consequently, if a master expects a reply that is dependent upon the byte it sent to a slave, it must ignore the first byte returned and transmit dummy bytes to receive subsequent reply bytes.

Table 10−5. SPI Interrupts Have Highest Priority and Jump to Address 0033h

Bit Name	Abbreviation	Name of Related SFR	Abbreviation	Address
Enable Auxiliary Interrupt	EAI	Enable Interrupt Control	EICON.5	D8h
Auxiliary Interrupt Flag	AI	Enable Interrupt Control	EICON.4	D8h
Enable SPI Transmit interrupt	ESPIT	Auxiliary Interrupt Enable	AIE.3	A6h
SPI Transmit Interrupt Status Flag	SPIT	Auxiliary Interrupt Status Register	AISTAT.3	A7h
	ESPIT		AIPOL.3	A4h
Enable SPI Receive Interrupt	ESPIR	Auxiliary Interrupt Enable	AIE.2	A6h
SPI Receiver Interrupt Status Flag	SPIR	Auxiliary Interrupt Status Register	AISTAT.2	A7h
	ESPIR		AIPOL.2	A4h

Table 10−6. PAI—Pending Auxiliary Interrupt Register

PAI		SFR A5h	Reset Value = 00h
Bit #	**Name**	**Interpretation When Read**	
7 − 4	−	Return 0	
3	PAI3	Auxiliary Interrupt Status 0000: No Pending Auxiliary IRQ 0001: Digital Low-Voltage or Hardware Breakpoint IRQ Pending 0010: Analog Low-Voltage IRQ Pending 0011: SPI Receive IRQ Pending or I^2C Status Interrupt Pending 0100: SPI Transmit IRQ Pending 0101: One Millisecond System Timer IRQ Pending 0110: Analog-to-Digital Conversion IRQ Pending 0111: Accumulator IRQ Pending 1000: One Second System Timer IRQ Pending	
2	PAI2		
1	PAI1		
0	PAI0		

10.4 SPI FIFO Buffer

If the FIFO buffer is to be used, its start and end addresses in core SRAM must be defined by writing to SPISTART at 9Eh and SPIEND at 9Fh, respectively. The SRAM between SPISTART and SPIEND (inclusive) should not be used by the application software.

The activity of the FIFO buffer is controlled by four pointers and two counters. Once initialized by writing to either SPICON or SPISTART, all pointers equal SPISTART and both counters are 0. Both SPISTART and SPIEND must be a location within SRAM between 80h and FEh. If a pointer to be incremented is equal to SPIEND, it will instead be set to SPISTART. The registers are changed as follows:

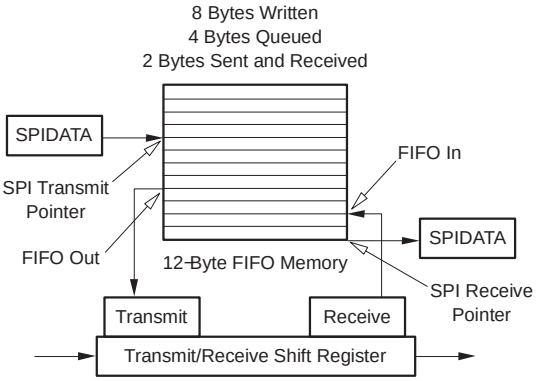
1) The CPU writes data to SPIDATA, which is copied to SRAM pointed to by CPUtxp. CPUtxp is then incremented along with TXcount. The CPU may write further bytes to SPIDATA during the following steps.

2) Txcount is no longer 0 and the byte pointed to by SPItxp is copied to TX BUF and on to the transmission shift register, TX SR. SPItxp is incremented and TXcount is decremented.

3) The synchronous nature of the SPI means that transmission of a byte always results in reception of a byte. This passes from the receiver shift register (RX SR), via RX BUF to the SRAM pointed to by SPIrxp. This SRAM location was used to hold a transmitted byte and is now overwritten with the received byte. SPIrxp is incremented along with RXcount.

4) The CPU reads data from SRAM pointed to by CPUrxp via SPIDATA. CPUrxp is incremented and RXcount is decremented.

Figure 10−3. SPI FIFO Operation



Special conditions:

1) If the FIFO is full when the CPU writes to SPIDATA, the data are discarded and neither CPUtxp nor TXcount are altered.

2) If the FIFO is empty and the CPU reads from SPIDATA, the value returned is undefined and neither CPUrxp nor RXcount are altered.

Table 10−7. SPISTART—SPI Buffer Start Address

SPISTART		SFR 9Eh Reset Value = 80h
Bit #	**Name**	**Action or Interpretation**
7	−	Always 1
6−0	SPISTART	Write: The start address of the circular FIFO buffer, somewhere within SRAM from 80h to FEh. The value must be less than SPIEND. The FIFO resides between SPISTART and SPIEND, inclusive. Writing to SPISTART initializes all the FIFO pointers and counters. CPUwrp =SPItxp = CPUrdp =SPIrxp = SPISTART, and TXcount = RXcount = 0.
	SPITP	Read: The current value of the CPU Transmit Pointer (CPUwrp). This is where the next byte for transmission is placed when the CPU writes to SPIDATA. Writing to SPIDATA increments the transmit counter and increments CPUwrp, unless that would make CPUtxp equal to the SPI Receive pointer (SPIrxp).

Table 10−8. SPISEND—SPI Buffer End Address

SPIEND		SFR 9Fh Reset Value = 80h
Bit #	**Name**	**Action or Interpretation**
7	−	Always 1
6−0	SPIEND	Write: The end address of the circular FIFO buffer, somewhere within SRAM from 80h to FFh. The value must be greater than SPISTART. The FIFO resides between SPISTART and SPIEND inclusive.
	SPIRP	Read: The current value of the CPU Receive Pointer (CPUrxp). This indicates where the next byte will be taken from as the CPU reads SPIDATA. Reading SPIDATA decrements the receive counter and increments the SPI Receive Pointer (SPIrxp) unless the receive counter is zero.

To sustain data transfers via the SPI while minimizing CPU overhead, the FIFO buffer will usually be filled and emptied in bursts by the application software. To coordinate this type of activity, the SPI Receive Control register (SPIRCON at 9Ch) and the SPI Transmit Control register (SPITCON at 9Dh) allow interrupts to be determined by the amount of data in the buffer. A receive interrupt can be set to occur when 2^N or more bytes have arrived, and a transmit interrupt when 2^N or less remain to be transmitted, where *N* is between 0 and 7.

The receive buffer may be flushed by writing a ‘1’ to RXFLUSH (bit 7 of SPIRCON), which causes CPUrxp to be set equal to SPIrxp, and RXcount to 0o.

The transmit buffer may be flushed by writing a ‘1’ to TXFLUSH (bit 7 of SPITCON), which causes CPUtxp to be set equal to SPItxp, and TXcount to 0.

Table 10−9. SPIRCON—SPI Receive Control Register

SPIRCON		SFR 9Ch Reset Value = 00h
Bit #	**Name**	**Action When Written**
7	RXFLUSH	Flush Receive FIFO. Write: 0: No effect. 1: The pointer used by the CPU to fetch data from the FIFO (CPUrxp) is set equal to the pointer used by the SPI interface to put data into the FIFO (SPIrxp). In effect, this clears the receive FIFO. The receive counter, RXcount, is also cleared.
6 − 3	−	Undefined
2	RXIRQ2	Receiver IRQ count threshold when in FIFO mode. RXIRQ = RXIRQ2:RXIRQ1:RXIRQ0 = 000_2 to 111_2 Generates SPI receive IRQ when receive count = 2^{RXIRQ} or more (that is, 1 or more to 128 or more). See ESPIR (bit 2 of AIE at A6h) and SPIR (bit 2 of AISTAT at A7h).
1	RXIRQ1	
0	RXIRQ0	
Bit #	**Name**	**Interpretation When Read**
7 − 0	RXCNT	The number of bytes in the FIFO and RX BUF still to be read (0 to 129). This is RXcount.

Table 10−10. SPITCON—SPI Transmit Control Register

SPITCON		SFR 9Dh Reset Value = 00h
Bit #	**Name**	**Action When Written**
7	TXFLUSH	Flush Transmit FIFO Write: 0: No effect 1: The pointer used by the CPU to put data into the FIFO (CPUtxp) is set equal to the pointer used by the SPI to get data from the FIFO (SPItxp). In effect, this clears the transmit FIFO. The transmit counter (TXcount) is also cleared.
6	−	Undefined
5	CLK_EN	SCK Driver Enable (in Master Mode) Write: 0: Disable SCK Driver 1: Enable SCK Driver
4	DRV_DLY	Drive Delay used with Drive Enable (DRV_EN). MOSI for master, MISO for slave. Write: 00: Disable (tri-state) immediately 01: Enable output drive immediately 10: Disable (tri-state) after current byte transfer 11: Enable output drive after current byte transfer
3	DRV_EN	
2	TXIRQ2	Transmit IRQ count threshold when in FIFO mode TXIRQ = TXIRQ2:TXIRQ1:TXIRQ0 = 000_2 to 111_2 Generates SPI transmit IRQ when transmit count = 2^{TXIRQ} or less (that is, 1 or less to 128 or less). See ESPIT (bit 3 of AIE at A6h) and SPIT (bit 3 of AISTAT at A7h).
1	TXIRQ1	
0	TXIRQ0	
Bit #	**Name**	**Interpretation When Read**
7−0	TXCNT	The number of bytes in the FIFO, TX BUF, and TX SR still to be transmitted (0 to 130). This is TXcount.

10.5 SPI Examples

Two examples are shown using the SPI. Example 10−1 shows a simple, polled environment.

Example 10−1. SPI, Simple, Polled Environment

```
// File SPIpolled.c - outputs and receives bytes via SPI
// MSC1211 EVM Switches 1:On  SW3-12345678 SW5-12345678
//                      0:Off     11110111     11110000
#include <Reg1211.h>
#include <stdio.h>
sbit RedLed = P3^4;         // RED LED on EVM
sbit YellowLed = P3^5;      // Yellow LED on EVM
sbit SlaveSelect = P1^0;    // avoids onboard SPI devices
code at 0xFFF3 void autobaud(void);

unsigned char SPIoutin(unsigned char n)
{
 while (!(AIE & 0x08)) RedLed=!RedLed;       //wait for TX empty
 SPIDATA=n;                 // output
 while (!(AIE & 0x04)) YellowLed=!YellowLed; //wait for RX
 return SPIDATA;            // input
}

void main(void)
{ data unsigned char i,j=0x41;
  AIE=0;                    // No interrupts
  PDCON&=~0x01;             // turns on SPI
  P1DDRH=0x40;              // CMOS output for P1.7
  P1DDRL=0x01;              // CMOS output for P1.4
  SPICON=0xC4;              // Divide 128, FIFO off, msb, master
  SPITCON=0x28;             // SCK driver on
  SlaveSelect=1;
  autobaud();
  printf("Simple polled (loopback) SPI\n");
  RI_0 = 0;                 // Clear received flag in UART
  while(1){
    while(!RI_0) {
      RedLed=YellowLed=0;
      SlaveSelect=0;
      i=SPIoutin(j);
      SlaveSelect=1;
      putchar(i);
    }
  RI_0 = 0;                 // any character to pause
  while(!RI_0);             // wait for character
  j=SBUF0;                  // get character
  RI_0 = 0;
  }                         // continue
}
```

A burst of characters is written to the FIFO in every second, and with MOSI and MISO joined together, they are read back by the CPU whenever the number of received characters is 16 or more. If the burst size is less than 16, 2 or more seconds will be needed to trigger a receive interrupt.

Example 10−2. SPI FIFO Mode

```
// File spiFIFOint.c – outputs and receives bytes via SPI FIFO
// MSC1211 EVM Switches 1:On  SW3–12345678 SW5–12345678
//                      0:Off     11110111     11110000
#include <Reg1211.h>
#include <stdio.h>
#define xtal 22118400
sbit RedLed = P3^4;          // RED LED on EVM
sbit YellowLed = P3^5;       // Yellow LED on EVM
sbit SlaveSelect = P1^0;     // avoids onboard SPI devices
code at 0xFFF3 void autobaud(void);

data unsigned char j=69;     // Letter 'E'

/* Auxiliary Interrupt */
void AuxInt(void) interrupt 6 using 1
{ unsigned char i;
  while (PAI) {
    switch(PAI) {
      case 3: // SPI RX
        {  while (SPIRCON) putchar(SPIDATA); //empty the buffer
           printf("\n");
           break; }
      case 8: // Seconds
        {  i=SECINT;         // remove seconds interrupt flag
           for(i=65;i<=j;i++) {
             while (!(AIE & 0x08));
             SPIDATA=i; }    // output
           break; }
    }
  }
  EICON&=~0x10;              // remove AI flag
}

void main(void)
{ PDCON&=~0x03;              // turns on System Timer and SPI
  P1DDRH=0x40;               // CMOS output for P1.7
  P1DDRL=0x01;               // CMOS output for P1.4
  SPIEND=0x9F;               // 32 byte buffer
  SPISTART=0x80;             // Start at 0x80 and initialise
  SPICON=0xD4;               // Divide 128, FIFO on, msb, master
  SPIRCON=0x04;              // RX IRQ on 16 or more
```

Example 10−2. (Continued)

```
  SPITCON=0x28;              // SCK driver on
  SlaveSelect=0;
  autobaud();
  MSEC=xtal/1000-1;          // 1ms tick
  HMSEC=100-1;               // 100ms tick
  SECINT=0x89;               // write 9 immediately for 10 x 100 ms
  printf("FIFO interrupt (loopback) SPI\n");
  AIE=0x84;                  // enable Seconds and SPI RX interrupts
  EICON|=0x20;               // enable auxiliary interrupt
  RI_0 = 0;                  // Clear received flag in UART
  while(1){
    while(!RI_0) {
      YellowLed=!YellowLed;  //main program
    }
  RI_0 = 0;                  // any character to pause
  while(!RI_0);              // wait for character
  j=SBUF0;                   // limit is received character A..a allowed
  RI_0 = 0;
  }                          // continue
}
```

Chapter 11

Timers and Counters

This chapter describes the MSC121x timers and counters.

Topic **Page**

11.1 Description 11-2
11.2 Timer/Counters 0 and 1 11-3
11.3 Timer/Counter 2 11-8
11.4 Example Program Using Timers 0, 1, and 2 11-13

11.1 Description

The MSC121x includes three Timer/Counter modules, 0 1 and 2, that behave in the same way as those found in the 8051/8052. When a module is clocked from the system clock it changes at a known rate and in this mode is referred to as a ‘Timer’. However, when clocked from an external source it may be considered as an event counter or timer. There are numerous modes of operation, which include but are not limited to:

- ❑ 13-bit timer
- ❑ 16-bit gated timer
- ❑ 16-bit gated counter
- ❑ 8-bit with auto reload
- ❑ 16-bit timer capture
- ❑ Baud rate generator

Note:

Not all modes are available within each module, but the combination of modes satisfies many application environments.

11.2 Timer/Counters 0 and 1

Bits in TMOD at 89h and TCON at 88h configure the operation of Timer/Counter 0 and Timer/Counter 1. They have identical relative behavior in modes 0, 1, and 2, but differ in mode 3, as expressed in the following tables and figures.

Table 11−1. TMOD—Timer Mode Control

TMOD		**SFR 89h** **Reset Value = 00h**
Bit #	**Name**	**Action or Interpretation**
7	Gate	Timer/Counter 1 Gate Control. Write: 0: Operation of Timer/Counter 1 does not depend upon pin P3.3/$\overline{\text{INT1}}$. 1: Pin P3.3/$\overline{\text{INT1}}$ has to be '1' to enable clocking. See TR1 (bit 6 of TCON at 88h).
6	C/$\overline{\text{T}}$	Timer/Counter 1 Select. Write: 0: Timer/Counter is clocked at f_{CLK}/12 (default) or f_{CLK}/4. See CKCON.4 at 8Eh. 1: Timer/Counter is clocked from pin P3.5/T1. See also TR1 (bit 6 of TCON at 88h).
5	M1	Timer/Counter 1 Mode Select. 00 (Mode 0): 13-bit counter 01 (Mode 1): 16-bit counter 10 (Mode 2): 8-bit counter with auto reload 11 (Mode 3): Timer/Counter 1 is halted, but holds its count. Same effect as clearing TR1.
4	M0	
3	Gate	Timer/Counter 0 Gate Control. Write: 0: Operation of Timer 1 does not depend upon pin P3.2/$\overline{\text{INT0}}$. 1: Pin P3.2/$\overline{\text{INT0}}$ has to be '1' to enable clocking. See TR0 (bit 4 of TCON at 88h).
2	C/$\overline{\text{T}}$	Timer/Counter 0 Select. Write: 0: Timer/Counter is clocked at f_{CLK}/12 (default) or f_{CLK}/4. See CKCON.3 at 8Eh. 1: Timer/Counter is clocked from pin P3.4 / T0. See TR0 (bit 4 of TCON at 88h).
1	M1	Timer/Counter 0 Mode Select. 00 (Mode 0): 13-bit counter 01 (Mode 1): 16-bit counter 10 (Mode 2): 8-bit counter with auto reload 11 (Mode 3): Timer/Counter 0 acts as two independent 8-bit Timer/Counters.
0	M0	

Table 11−2. TCON—Timer/Counter Control

TCON		SFR 88h Reset Value = 00h
Bit #	**Name**	**Action or Interpretation**
7	TF1	Timer 1 (Interrupt) Overflow Flag. Read: 0: No Overflow. 1: Timer 1 reached the maximum count and changed to 0. Write: 0: Clear flag. 1: Set flag and generate interrupt request if unmasked. Cleared in software by writing 0, or cleared automatically as the processor jumps to the interrupt service routine at 001Bh.
6	TR1	Timer 1 Run Control. Write: 0: Timer 1 cannot be clocked. 1: Timer 1 may be clocked.
5	TF0	Timer 0 (Interrupt) Overflow Flag. Read: 0: No Overflow. 1: Timer 0 reached maximum count and changed to 0. Write: 0: Clear flag. 1: Set flag and generate interrupt request if unmasked. Cleared in software by writing 0, or cleared automatically as the processor jumps to the interrupt service routine at 000Bh.
4	TR0	Timer 0 Run Control. Write: 0: Timer 0 cannot be clocked. 1: Timer 0 may be clocked.
3	IE1	Interrupt 1 Edge Detect. If $\overline{\text{INT1}}$ is edge-sensitive because IT1 = 1, IE1 is set when a negative edge is detected. It is cleared when the CPU jumps to the interrupt service routine at 0013h or by writing 0 in software. If IT1 = 0, IE1 is set when the $\overline{\text{INT1}}$ pin is low, and cleared when the $\overline{\text{INT1}}$ pin is high.
2	IT1	Interrupt 1 type select. Write: 0: $\overline{\text{INT1}}$ is sensitive to a low level. 1: $\overline{\text{INT1}}$ is sensitive to a negative (falling) edge.
1	IE0	Interrupt 0 edge detect. If $\overline{\text{INT0}}$ is edge-sensitive because IT0 = 1, IE0 is set when a negative edge is detected. It is cleared when the CPU jumps to the interrupt service routine at 0013h or by writing 0 in software. If IT0 = 0, IE1 is set when the $\overline{\text{INT0}}$ pin is low, and cleared when the $\overline{\text{INT0}}$ pin is high.
0	IT0	Interrupt 0 type select. Write: 0: $\overline{\text{INT0}}$ is sensitive to a low level. 1: $\overline{\text{INT0}}$ is sensitive to a negative (falling) edge.

NOTE: Bit 0 to bit 3 of TCON are not associated with the operation of any Timer/Counter.

11.2.1 Modes 0 and 1

The description that follows is with respect to Timer/Counter 0. but also applies to Timer/Counter 1 with appropriate re-allocation of control bits. However, the overflow condition of Timer 1 alone is able to act as a reference clock for the serial ports.

TH0:TL0 represents a 13-bit, negative-edge triggered up counter that can be clocked from a variety of sources. When C/$\overline{T}$ is 0, it behaves as a gated timer running at either f_{CLK}/12 (default) or f_{CLK}/4. However, when C/$\overline{T}$ is 1, it behaves as a gated event counter, where appropriate transitions on pin T0, TR0, GATE, or pin $\overline{INT0}$ cause it to increment.

In mode 0, the upper three bits of TL0 are undefined and should not be used.

When TH0 overflows from FFh to 00h, the interrupt flag (TF0) is set. It is cleared automatically as the CPU jumps to the service routine at 000Bh, or cleared manually by writing a ‘0’ to it in software.

Figure 11−1. Timer 0/1—Modes 0 and 1

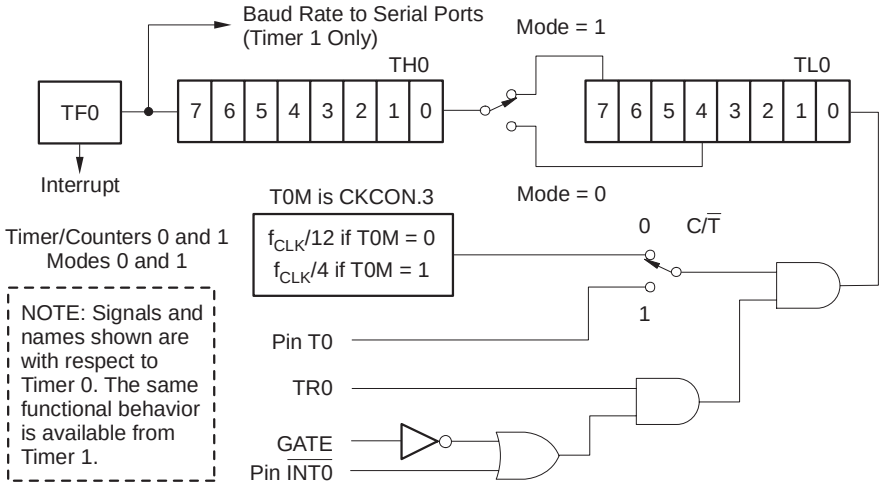


Table 11−3. Modes 0 and 1 Operation

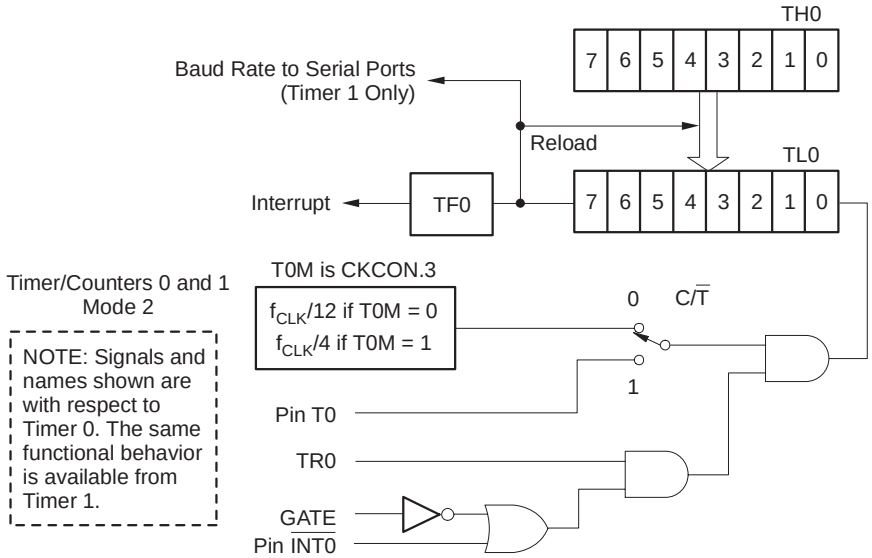
C/$\overline{T}$	T0M	Pin T0	TR0	GATE	Pin $\overline{INT0}$	CLOCK
0	0	x	1	0	x	f_{CLK}/12
0	0	x	1	1	1	f_{CLK}/12
0	1	x	1	0	x	f_{CLK}/4
0	1	x	1	1	1	f_{CLK}/4
1	x	1 to 0	1	0	x	Increment
1	x	1 to 0	1	1	1	Increment
1	x	1	1 to 0	0	x	Increment
1	x	1	1 to 0	1	1	Increment
1	x	1	1	0 to 1	0	Increment
1	x	1	1	1	1 to 0	Increment

NOTE: For all other combinations of control bits and pins, TH0:TL0 is unchanged.

11.2.2 Mode 2

The description that follows is with respect to Timer/Counter 0, but applies to Timer/Counter 1 with appropriate re-allocation of control bits. However, the overflow condition of Timer 1 alone is able to act as a reference clock for the serial ports.

Figure 11−2. Timer 0/1—Mode 2

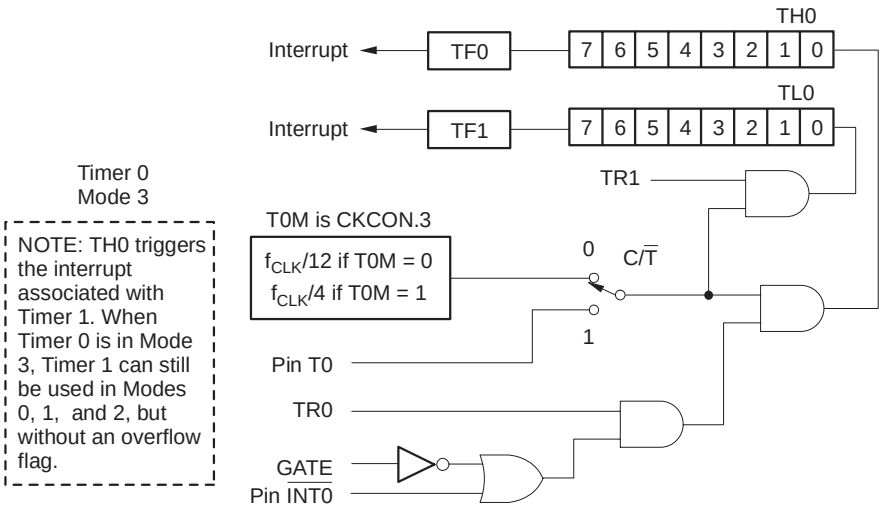


TL0 represents an 8-bit, negative-edge triggered counter that is reloaded from TH0 as it overflows. It may be from clocked from a variety of sources as described for modes 0 and 1.

11.2.3 Mode 3

The behavior of Timer/Counter 0 in mode 3 is not the same as Timer/Counter 1 because the interrupt flag usually associated with Timer 1 is controlled by TH0.

Figure 11−3. Timer 0—Mode 3



TL0 is an 8-bit timer or counter that is clocked and gated in a manner similar to Mode 0. TH0 must be clocked from the same source but is gated only by control bit TR1. Without TR1 and TF1, Timer 1 can still be used for baud rate generation.

11.2.4 Summary of Control Bits and SFRs for Timer/Counters 0 and 1

Table 11−4. Control Bit and SFR Summary for Timer/Counters 0 and 1

Signal, Control or Data		Timer 1			Timer 0		
		Name or Bit	SFR Address	SFR Bit Address	Name or Bit	SFR Address	SFR Bit Address
Timer overflow flag	TFx	TCON.7	88h	8Fh	TCON.5	88h	8Dh
Count high byte		TH1	8Dh		TH0	8Ch	
Count low byte		TL1	8Bh		TL0	8Ah	
Timer/Counter select	C/$\overline{T}$	TMOD.6	89h		TMOD.2	89h	
Mode bit 1	M1	TMOD.5	89h		TMOD.1	89h	
Mode bit 0	M0	TMOD.4	89h		TMOD.0	89h	
Divide by 4 or 12 select	TxM	CKCON.4	8Eh		CKCON.3	8Eh	
External clock input	Tx	P3.5/T1	B0h	B5h	P3.4/T0	B0h	B4h
Timer run control	TRx	TCON.6	88h	8Eh	TCON.4	88h	8Ch
Internal timer gate	GATE	TMOD.7	89h		TMOD.3	89h	
External timer gate	$\overline{INTx}$	P3.3/$\overline{INT1}$	B0h	B3h	P3.2/$\overline{INT0}$	B0h	B2h
Enable interrupt	ETx	IE.3	A8h	ABh	IE.1	A8h	A9h
Interrupt priority	PTx	IP.3	B8h	BBh	IP.1	B8h	B9h

11.3 Timer/Counter 2

Timer/Counter 2 consists of the register pair TH2:TL2, which act as a 16-bit, negative-edge triggered up counter. The associated register pair, RCAPH:RCAPL, may either capture the current value of TH2:TL2 or provide a reload value according to the mode of operation selected by bits in T2CON at C8h.

Table 11−5. T2CON—Timer 2 Control

T2CON		**SFR C8h** **Reset Value = 00h**
Bit #	**Name**	**Action or Interpretation**
7	TF2	Timer 2 Overflow Flag. Read: 0: No Overflow. 1: Timer 2 reached maximum count of FFFFh and overflowed to 0. It is *not* cleared automatically as the processor jumps to the interrupt service routine at 002Bh. Write: 0: Clear flag if set. 1: Set overflow flag and generate interrupt if enabled.
6	EXF2	Timer 2 External Flag. This flag is set by a high-to-low transition on pin P1.1/T2EX with EXEN2 previously set. Write: 0: Clear flag if set. 1: Set overflow flag and generate interrupt if enabled.
5	RCLK	Receive Clock Select. Write: 0 (or 1): Timer 1 (or 2) overflow rate determines the receiver baud rate for USART0 in modes 1 or 3. Setting this bit forces Timer 2 into a 16-bit auto-reload mode where the reference clock is $f_{CLK}/2$ or pin P1.0/T2. USART 1 can only be clocked from Timer/Counter 1.
4	TCLK	Transmit Clock Select. Write: 0 (or 1): Timer 1 (or 2) overflow rate determines the transmitter baud rate for USART0 in serial modes 1 or 3. Setting this bit forces Timer 2 into a 16-bit auto-reload mode where the reference clock is $f_{CLK}/2$ or pin P1.0/T2. USART 1 can only be clocked from Timer/Counter 1.
3	EXEN2	Timer 2 External Enable. Write: 0: Ignore negative edges on pin P1.1/T2EX. 1: Negative edge on pin P1.1/T2EX sets EXF2 and causes capture or reload depending on the operating mode of Timer/Counter.
2	TR2	Timer 2 Run Control. Write: 0: Timer 2 cannot be clocked. 1: Timer 2 may be clocked.
1	C/$\overline{T2}$	Timer 2 Counter/Timer Select. Write: 0: Counter/Timer is clocked at $f_{CLK}/12$ (default) or $f_{CLK}/4$; or $f_{CLK}/2$ in Baud Rate mode. 1: Counter/Timer is clocked from pin P1.0/T2.

Table 11−5. T2CON—Timer 2 Control (continued)

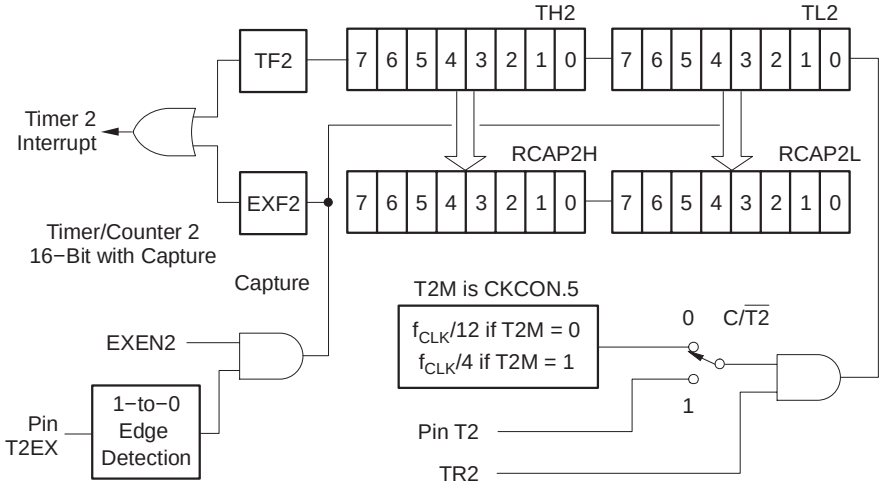
T2CON		SFR C8h					Reset Value = 00h
Bit #	**Name**	**Action or Interpretation**					
0	CP/$\overline{RL2}$	Capture/Reload Select.					
		TR2	**RCLK**	**TCLK**	**CP/$\overline{RL2}$**	**Mode**	**TF2 Overflow**
		0	x	x	x	Hold	
		1	0	0	0	16-bit Timer/Counter with auto-reload	Yes
		1	0	0	1	16-bit Timer/Counter with capture	Yes
		1	1	x	x	Baud Rate Generator	No
		1	x	1	x		

11.3.1 16-Bit Timer/Counter with Optional Capture

To select this mode, RCLK, TCLK, and CP/$\overline{RL2}$ in T2CON must all be 0.

Control bit TR2 is active high and enables either an internal clock or an external clock on pin P1.0/T2, according to the state of C/$\overline{T2}$. Specifically, when C/$\overline{T2}$ is 0, TH2:TL2 is a gated timer running at either f_{CLK} /12 (default) or f_{CLK}/4. However, when C/$\overline{T2}$ is 1, it is a gated event counter.

Figure 11−4. Timer/Counter 2—16-Bit with Capture



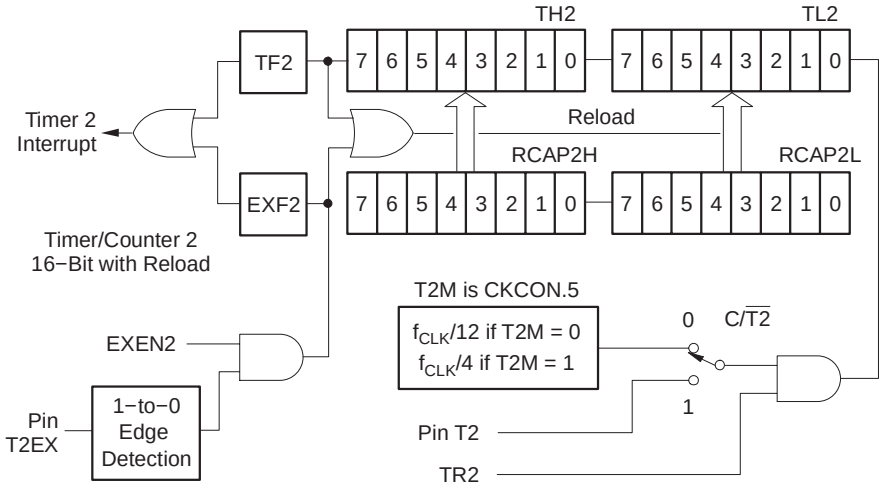
As TH2:TL2 overflows from FFFFh to 0000h, the interrupt flag (TF2) is set; this must be cleared in software. If interrupt enables ET2 and EA (bits 5 and 7, respectively, of IE at A8h) are both 1, the CPU jumps to the interrupt service routine at 002Bh. Writing a ‘1’ to TF2 causes an interrupt, if it is enabled.

A negative-edge on pin P1.1/T2EX when control bit EXEN2 is 1 causes the current value of TH2:TL2 to be copied into capture registers RCAP2H:RCAP2L and sets the interrupt flag (EXF2). This is ORed with TF2 and may cause a Timer2 interrupt in a manner similar to TF2. EXF2 has to be cleared in software and writing a ‘1’ to it causes an interrupt, if it is enabled.

11.3.2 16-Bit Timer/Counter with Automatic and Forced Reload

When RCLK and TCLK are both 0 but CP/$\overline{RL2}$ is 1, RCAP2H:RCAP2L does not contain a captured value as in the previous mode. Instead, it represents the value to be reloaded into TH2:TL2. Reloading occurs because either TH2:TL2 overflows from FFFFh to 0000h, or pin P1.1/T2EX changes from 1 to 0, while EXEN2 is 1.

Figure 11−5. Timer/Counter 2—16-Bit with Reload



Control bit TR2 is active high and enables either an internal clock or an external clock on pin P1.0/T2, according to the state of $\overline{C/T2}$. Specifically, when C/$\overline{T2}$ is 0, TH2:TL2 is a gated timer running at either $f_{CLK}/12$ (default) or $f_{CLK}/4$. However, when C/$\overline{T2}$ is 1, it is a gated event counter.

As TH2:TL2 overflows from FFFFh to 0000h, the interrupt flag (TF2) is set; this must be cleared in software. If interrupt enables ET2 and EA (bits 5 and 7, respectively, of IE at A8h) are both 1, the CPU jumps to the interrupt service routine at 002Bh. Writing a ‘1’ to TF2 causes an interrupt if it is enabled.

A negative-edge on pin P1.1/T2EX when control bit EXEN2 is 1, causes the interrupt flag (EXF2) to be set. It is ORed with TF2 and may cause a Timer2 interrupt in a manner similar to TF2. EXF2 has to be cleared in software; writing a ‘1’ to it causes an interrupt, if it is enabled.

11.3.3 Baud Rate Generator

When either RCLK is 1 or TCLK is 1, Timer/Counter 2 operates as a baud rate generator for serial port 0. In this mode, TH2:TL2 is reloaded from RCAP2H:RCAP2L whenever it overflows from FFFFh to 0000h.

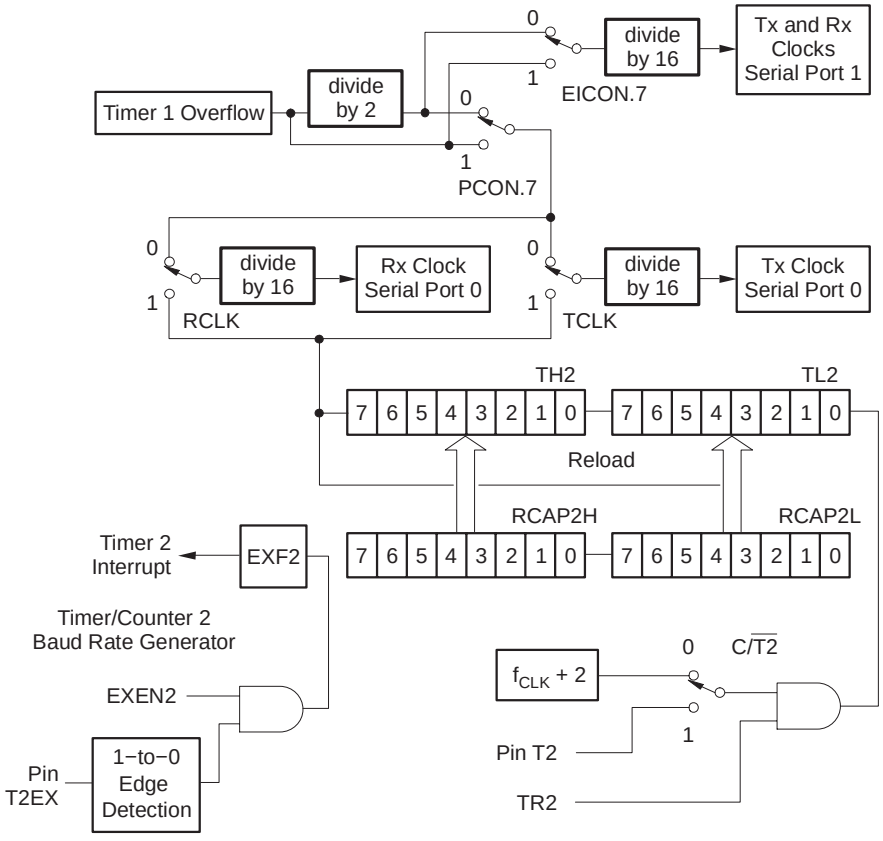
Control bit TR2 is active high and enables either an internal clock or an external clock on pin P1.0/T2 according to the state of C/$\overline{T2}$. Specifically, when C/$\overline{T2}$ is 0, TH2:TL2 runs at $f_{CLK}/2$, otherwise when C/$\overline{T2}$ is 1, it runs at a rate determined by pin P1.0/T2.

A negative-edge on pin P1.1/T2EX when control bit EXEN2 is 1 causes the interrupt flag (EXF2) to be set. If interrupt enables ET2 and EA (bits 5 and 7, respectively, of IE at A8h) are both 1, the CPU jumps to the interrupt service routine at 002Bh. EXF2 has to be cleared in software and writing a '1' to it causes an interrupt, if enabled.

To accommodate applications that require different transmit and receive baud rates, the overflow of Timer 1, optionally divided by 2, may be selected as shown in Figure 11−6.

When Timer/Counter 2 uses the internal clock to determine the baud rate of serial port 0, the rate is given by $f_{CLK}/32/(65536 - RCAP2H:RCAP2L)$.

Figure 11−6. Timer/Counter 2—Baud Rate Generator



11.3.4 Summary of Control Bits and SFRs for Timer/Counter 2

Table 11−6. Control Bit and SFR Summary for Timer/Counter 2

Signal, Control, or Data		Timer 2		
		Name or Bit	SFR Address	SFR Bit Address
Timer overflow flag	TF2	T2CON.7	C8h	CFh
External interrupt flag	EXF2	T2CON.6	C8h	CEh
Count high byte		TH2	CDh	
Count low byte		TL2	CCh	
Capture/reload high byte		RCAP2H	CBh	
Capture/reload low byte		RCAP2L	CAh	
Timer/counter select	C/$\overline{T2}$	T2CON.1	C8h	C9h
Receiver clock select	RCLK	T2CON.5	C8h	CDh
Transmitter clock select	TCLK	T2CON.4	C8h	CCh
Capture/reload flag	CP/$\overline{RL2}$	T2CON.0	C8h	C8h
Divide by 4 or 12 select	T2M	CKCON.5	8Eh	
External clock input	T2	P1.0 / T2	90h	90h
Timer run control	TR2	T2CON.2	C8h	CAh
Internal timer gate	EXEN2	T2CON.3	C8h	CBh
External trigger input	T2EX	P1.1	90h	91h
Enable interrupt	ET2	IE.5	A8h	ADh
Interrupt priority	PT2	IP.5	B8h	BDh

11.4 Example Program Using Timers 0, 1, and 2

In the example, the red and yellow LEDs on the MSC1210 evaluation module are associated with overflow interrupts from Timers 0 and 2, respectively. Serial port 0 is repeatedly tested for receipt of any character at a baud rate of 9600, as determined by Timer 1. The character is echoed and TF2 is set to generate an additional interrupt.

The red LED is on for 1 second and then off for 1 second, while the yellow LED flashes at half this rate. Initially, the LEDs light at the same moment, but each received character causes an extra call (Timer2Int), which produces an increasing visible phase shift between the flashing of the LEDs.

f_{CLK} is the same frequency as the external (crystal) oscillator, unless the System Clock Divider SFR (SYSCLK at C7h) is present and active. SYSCLK is provided in the MSC1211/12/13/14 with a default value that causes no division of the external clock.

Table 11−7. Timer Modes

Timer	Mode	Type	Clock	Overflow or Baud Rate (11.0592MHz Clock)
0	0	13-bit	$f_{CLK}/12$	$\frac{1}{12} \times \frac{f_{CLK}}{8192} = \frac{11059200}{98304} = 112.5$
1	2	8-bit reload	$f_{CLK}/12$	$\frac{1}{2} \times \frac{1}{16} \times \frac{f_{CLK}}{12 \times (256 - TH1)} = \frac{11059200}{1152} = 9600$ Baud Where TH1 = 253
2		16-bit reload	$f_{CLK}/4$	$\frac{1}{4} \times \frac{f_{CLK}}{(65536 - RCAP2H:RCAP2L)} = \frac{11059200}{4 \times 12288} = 225$ Where RCAP2H:RCAP2L = 53248

Example 11−1. Program Using Timers 0, 1, and 2

```
// File Timer012b.c – Timer 0 in 13–bit mode
// Timer 1 in 8–bit reload and Timer 2 in releoad
// MSC1210 EVM Switches 1:On  SW3–12345678 SW6–12345678
//                      0:Off     11110111     11110000
#include <Reg1210.h>
#include <stdio.h>
#define fclk 11059200
#define BAUD 9600
#define limit 225
sbit RedLed    = P3^4;      // RED LED on EVM
sbit YellowLed = P3^5;      // Yellow LED on EVM

data unsigned int i=1, j=1;

void Timer0Int(void) interrupt 1 using 1
{ if(!--i) {i=limit; YellowLed=!YellowLed;}
}

void Timer2Int(void) interrupt 5 using 1
{ if(!--j) {j=limit; RedLed=!RedLed;}
  TF2=0;                    // remove overflow flag
}

void main(void)
{ CKCON=0x20;               // Timer 2 at fclk/4; Timers 0,1 at fclk/12
  PCON =0x30;               // SMOD = 0
  TMOD =0x20;               // Timer 1 Auto reload; Timer 0 13–bit
  TCON =0x50;               // TR1 and TR0 are 1
  TH1  =256–fclk/32/12/BAUD; // Timer 1 reload value
  T2CON=0x04;               // TR2 is 1, and Timer 2 is auto-reload
  RCAP2=65536–fclk/4/limit;
  SCON0=0x52;               // Asynchronous and enabled, TI_0=1, RI_0=0
  while(!RI_0);             // wait for key press
  printf("\nMSC1210 Timer 0 in mode 0, Timer 1 in mode 2");
  printf("\n        Timer 2 in 16–bit auto reload\n");
  IE   =0xA2;               // EA ET2 and ET1 enabled
  while(1){
    while(!RI_0);           // wait for key press
    SBUF0=SBUF0;            // echo
    TF2=1;                  // Force Timer 2 interrupt
    RI_0=0;
    }
}
```

Chapter 12

Serial Ports (USART0 and USART1)

This chapter describes the serial ports of the MSC121x.

Topic **Page**

12.1 Description 12-2
12.2 Control Bits in SCON0 and SCON1 12-3
12.3 Pin and Interrupt Assignments 12-4
12.4 Timer/Counters 1 and 2 and Baud Rate Generation 12-5
12.5 Mode 0—8-Bit Synchronous 12-7
12.6 Mode 1—10-Bit Asynchronous 12-8
12.7 Mode 2 and 3—11-Bit Asynchronous 12-9
12.8 Multiprocessor Communications 12-10
12.9 Example Program 12-10

12.1 Description

The MSC121x has two serial ports. Both may be configured in almost the same variety of synchronous or asynchronous modes and clocked via f_{CLK}, the overflow from Timer/Counter 1 or Timer/Counter 2. USART stands for *Universal Synchronous/Asynchronous Receiver/Transmitter.*

Each port has a control register and a data register, referenced as SCON0 at 98h and SBUF0 at 99h for serial port 0 and as SCON1 at C0h and SBUF1 at C1h for serial port 1.

12.2 Control Bits in SCON0 and SCON1

Table 12−1. SCON0 and SCON1—Serial Port 0 and Serial Port 1 Control

SCON0			SFR 98h						Reset Value = 00h
SCON1			SFR C0h						
			Action or interpretation						
Bit #	Name0	Name1	Mode	SM0	SM1	SM2	Function	Length	Rate (1)
7	SM0_0	SM0_1	0	0	0	0	Synchronous	8	f_{CLK} / 12
			0	0	0	1	Synchronous	8	f_{CLK} / 4
			1	0	1	0	Asynchronous	10	Timer (2)
						1 (3)	Asynchronous		
6	SM1_0	SM1_1	2	1	0	0	Asynchronous	11	$\frac{2^{SMOD}}{64} \times f_{CLK}$ (4)
						1 (5)	Asynchronous (Multiprocessor)	11	
5	SM2_0	SM2_1	3	1	1	0	Asynchronous	11	Timer (2)
			3	1	1	1 (5)	Asynchronous (Multiprocessor)	11	Timer (2)
4	REN_0	REN_1	Receive Enable. Write: 0: receive shift register is disabled 1: receive shift register is enabled (for mode 0, RI = 0 is also required).						
3	TB8_0	TB8_1	9th Transmission Bit State. The state of the 9th bit to be transmitted in modes 2 and 3.						
2	RB8_0	RB8_1	9th Received Bit State. The state of the 9th bit received in modes 2 and 3. In mode 1, when SM2 = 0, RB8_0 is the state of the stop bit. RB8_0 is not used in mode 0.						
1	TI_0	TI_1	TI_0 Transmitter Interrupt Flag. This bit is set when the transmit buffer has been completely shifted out. In mode 0, this occurs at the end of the 8th data bit, while in all other modes it is set at the beginning of the STOP bit. This flag must be manually cleared by software and can be set in software to cause an interrupt.						
0	RI_0	RI_1	RI_0 Receiver Interrupt Flag. This bit indicates that a byte has been received in the input shift register. In mode 0, it is set at the end of the 8th data bit; in mode 1, after the last sample of the incoming stop bit; and in modes 2 and 3, after the last sample of the 9th data bit. This bit must be manually cleared by software and can be set in software to cause an interrupt.						

1) If IDLE (bit 7 of PCON at 87h) is set, the CPU, Timer/Counters 0,1 and 2, and both serial ports will freeze until there is an auxiliary interrupt or external wake-up (see AIE at A6h, EICON at D8h and EWU at C6h).

2) In modes 1 and 3, serial port 0 may be clocked by Timer/Counter 1 or Timer/Counter 2, as determined by RCLK and TCLK (see T2CON at C8h); whereas serial port 1 may be clocked only by Timer/Counter 1.

3) In mode 1 with SM2 = 1, RI is activated only if a valid stop bit is received.

4) For USART0, SMOD0 is bit 7 of PCON at 87h. For USART1, SMOD1 is bit 7 of EICON at D8h.

5) In modes 2 and 3 with SM2 = 1, RI is activated only if the ninth received data bit is 1.

12.3 Pin and Interrupt Assignments

Table 12−2. USART Pin and Interrupt Assignments

Function	USART 0 : SBUF0 at 99h			USART 1 : SBUF1 at C1h		
	Rx Pin	**Tx Pin**	**Clock**	**Rx Pin**	**Tx Pin**	**Clock**
Mode 0 Transmit triggered by write to SBUF		P3.0	P3.1		P1.2	P1.3
Mode 0 Receive triggered by REN = 1 and RI = 0	P3.0		P3.1	P1.2		P1.3
Modes 1, 2, and 3 Transmit triggered by write to SBUF with TI = 1 Received data read via SBUF when RI = 1 and REN = 1.	P3.0	P3.1	−	P1.2	P1.3	−
	Name	**SFR Address**	**SFR Bit Address**	**Name**	**SFR Address**	**SFR Bit Address**
Interrupt Enable	IE.4	A8h	ACh	IE.6	A8h	AEh
Interrupt Priority	IP.4	B8h	BCh	IP.6	B8h	BEh

NOTE: In mode 0, the Rx pin is used to receive and transmit synchronous data. Consequently, the corresponding data direction bits should be defined as input, output, or bidirectional, as appropriate.

12.4 Timer/Counters 1 and 2 Baud Rate Generation

In asynchronous modes 1 and 3, the overflow rate of either Timer/Counter 1 or Timer/Counter 2 can determine the receive or transmit baud rate for serial port 0. However, in these modes, USART1 can only use the overflow rate of Timer/Counter 1.

The overflow of all Timer/Counters passes through a fixed divide-by-16 counter (see Figure 11−6) that is reset when a START condition is identified. By default, the overflow output of Timer/Counter 1 is also divided by 2, but this may be avoided if SMODx = 1.

For Timer/Counter 1, see Table 12−4.

Table 12−3. Timer/Counter 2 Baud Rate Generation

Configuration Bits in T2CON at C8h		**Serial port 0 Rx Baud Rate**		**Serial port 0 Tx Baud Rate**		**Serial port 1 Rx and Tx Baud Rate**	
RCLK bit 5	TCLK bit 4	SMOD0 = 0	SMOD0 = 1	SMOD0 = 0	SMOD0 = 1	SMOD1 = 0	SMOD1 = 1
0	0	$\frac{\text{Timer 1}}{32}$	$\frac{\text{Timer 1}}{16}$	$\frac{\text{Timer 1}}{32}$	$\frac{\text{Timer 1}}{16}$	$\frac{\text{Timer 1}}{32}$	$\frac{\text{Timer 1}}{16}$
0	1	$\frac{\text{Timer 1}}{32}$	$\frac{\text{Timer 1}}{16}$	$\frac{\text{Timer 2}}{16}$	$\frac{\text{Timer 2}}{16}$	$\frac{\text{Timer 1}}{32}$	$\frac{\text{Timer 1}}{16}$
1	0	$\frac{\text{Timer 2}}{16}$	$\frac{\text{Timer 2}}{16}$	$\frac{\text{Timer 1}}{32}$	$\frac{\text{Timer 1}}{16}$	$\frac{\text{Timer 1}}{32}$	$\frac{\text{Timer 1}}{16}$
1	1	$\frac{\text{Timer 2}}{16}$	$\frac{\text{Timer 2}}{16}$	$\frac{\text{Timer 2}}{16}$	$\frac{\text{Timer 2}}{16}$	$\frac{\text{Timer 1}}{32}$	$\frac{\text{Timer 1}}{16}$
		SMOD0 is bit 7 of PCON at 87h				SMOD1 is bit 7 of EICON at D8h	

Table 12−4. Timer/Counter 1 Baud Rate Generation

Mode of Timer/Counter 1 Determined by TMOD at 89h		**Timer/Counter 1 Overflow Rate When Clocked Internally TMOD.6 = 0**		**Timer/Counter 1 Overflow Rate When Clocked Externally TMOD.6 = 1**
M1 (bit 5)	M0 (bit 4)	CKCON.4 = T1M = 0	CKCON.4 = T1M = 1	CKCON.4 = T1M = 0 or 1
0	0	$\frac{f_{CLK}}{12 \times 8192}$	$\frac{f_{CLK}}{4 \times 8192}$	$\frac{f_{T1}}{8192}$
0	1	$\frac{f_{CLK}}{12 \times 65536}$	$\frac{f_{CLK}}{12 \times 655362}$	$\frac{f_{T1}}{65536}$
1	0	$\frac{f_{CLK}}{12 \times (256 - TH1)}$	$\frac{f_{CLK}}{4 \times (256 - TH1)}$	$\frac{f_{T1}}{256 - TH1}$
1	1	Stopped	Stopped	Stopped

NOTES: f_{T1} is the frequency of the signal at pin P3.5/T1.

For Timer/Counter 2, the overflow rate is:

when clocked internally because T2CON.1 = 0, Overflow Rate $= \frac{f_{CLK}}{2 \times (65536 - RCAP2H:RCAP2L)}$

when clocked from pin P1.0/T2 because T2CON.1 = 1, Overflow Rate $= \frac{f_{T2}}{(65536 - RCAP2H:RCAP2L)}$

Table 12−5. USART Baud Rate Generation

Serial Port #	Timer #	Conditions	Baud Rate
0	1	T2CON = $xx00xxxx_2$; PCON.7 = SMOD0 = 1; TCON = 01000000_2; TMOD = $0100xxxx_2$; f_{T1} =19.660800MHz.	$= \frac{1}{16} \times \frac{f_{T1}}{8192} = \frac{19660800}{131072} = 150$
0	1	T2CON = $xx00xxxx_2$; PCON.7 = SMOD0 = 0; CKCON.4 = T1M = 0; TCON = 01000000_2; TMOD = $0010xxxx_2$; TH1 = 253; f_{CLK} = 11.059200MHz	$= \frac{1}{2} \times \frac{1}{16} \times \frac{f_{CLK}}{12 \times (256 - TH1)} = \frac{11059200}{384 \times 3} = 9600$
1	1	T2CON = $xx00xxxx_2$; EICON.7 = SMOD1 = 1; CKCON.4 = T1M = 1; TCON = 01000000_2; TMOD = $0010xxxx_2$; TH1 = 112; f_{CLK} = 11.059200MHz	$= \frac{1}{16} \times \frac{f_{CLK}}{4 \times (256 - TH1)} = \frac{11059200}{64 \times 144} = 1200$
1	1	T2CON = $xx00xxxx_2$; EICON.7 = SMOD1 = 1; TCON = 01000000_2; TMOD = $0110xxxx_2$; TH1 = 184; f_{T1} = 22.118400MHz	$= \frac{1}{16} \times \frac{f_{T1}}{(256 - TH1)} = \frac{22118400}{16 \times 72} = 19200$
0	2	T2CON = 00110100_2; RCAP2H:RCAP2L = 65211; f_{CLK} = 25MHz	$= \frac{1}{2} \times \frac{1}{16} \times \frac{f_{CLK}}{(65536 - RCAP2H:RCAP2L)}$ $= \frac{25000000}{32 \times 325} = 2404$
0	2	T2CON = 00110110_2; RCAP2H:RCAP2L = 65406; f_{T2} = 10MHz	$= \frac{1}{2} \times \frac{1}{16} \times \frac{f_{T2}}{(65536 - RCAP2H:RCAP2L)}$ $= \frac{10000000}{16 \times 130} = 4808$

12.5 Mode 0—8-Bit Synchronous

In mode 0, serial data is either received or transmitted eight bits at a time in a synchronous fashion with respect to a shared input/output pin and a common clock output. Reception is triggered when REN = 1 and RI = 0, while transmission is triggered by a write to SBUF. If SM2 is 0, the clock runs at $f_{CLK}/12$; otherwise, it runs at $f_{CLK}/4$, as shown in Figure 12−1 and Figure 12−2. There are no start or stop bits in this mode.

RI is set three f_{CLK} cycles after the eighth bit has been received, and TI is set three f_{CLK} cycles after the eighth bit has been transmitted. This is true when SM2 = 0, but the delays change to four f_{CLK} cycles when SM2 = 1.

Figure 12−1. Synchronous Receive at $f_{CLK}/4$

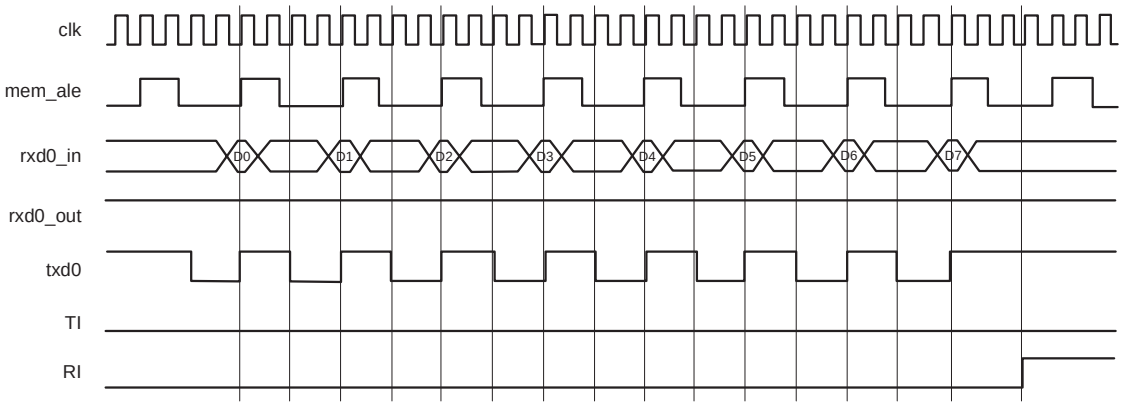
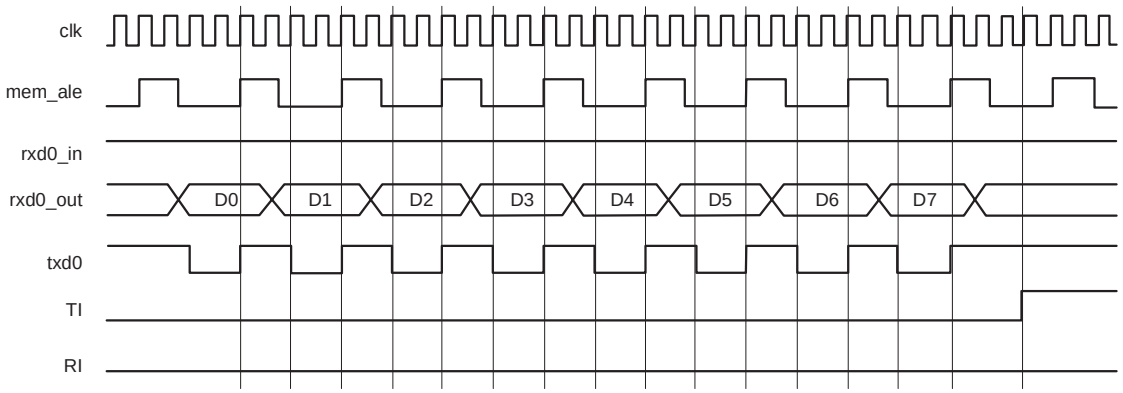


Figure 12−2. Synchronous Transmit at $f_{CLK}/4$



12.6 Mode 1—10-Bit Asynchronous

In mode 1, serial data is received or transmitted eight bits at a time, in an asynchronous fashion with respect to independent input and output pins. The baud rate is determined by the overflow rate of Timer/Counter 1 or 2 for serial port 0, or Timer/Counter 1 for serial port 1. Reception of a byte begins when REN is 1 and a start bit is recognized. This occurs after a high-to-low transition on the receive pin, followed by a low level on two of three consecutive samples made at 7/16th, 8/16th, and 9/16th of the bit time. In this way, short-lived pulses are not regarded as a valid start bit. During reception, eight bits are shifted into an input shift register, which is then loaded into the received SBUF register if:

1) RI is 0, and

2) SM2 is 1 and the stop bit is 1, or SM2 is 0 (that is, the state of the stop bit does not matter).

If these conditions are not met, the received data is lost and RI is not set. If SBUF is loaded, the state of the stop bit is copied into RB8.

Transmission is triggered by a write to SBUF and results in a 10-bit frame consisting of a low-level start bit, eight data bits, and a high-level stop bit. The start bit begins at the next rollover of the local divide-by-16 counter, and TI is set at the beginning of the stop bit.

Figure 12−3. Asynchronous 10-Bit Transmit Timing

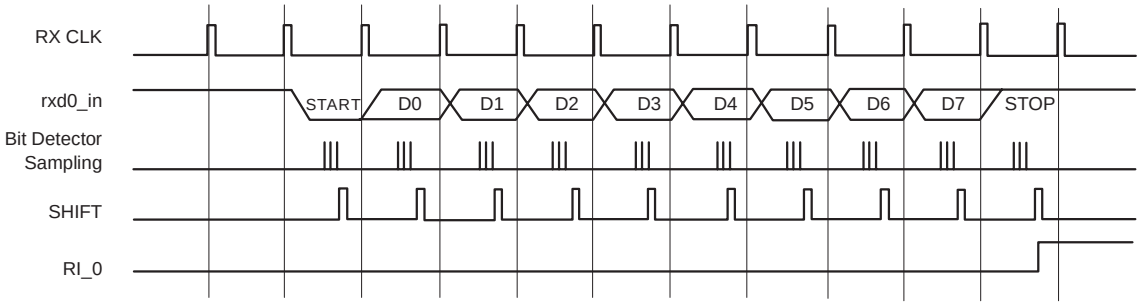
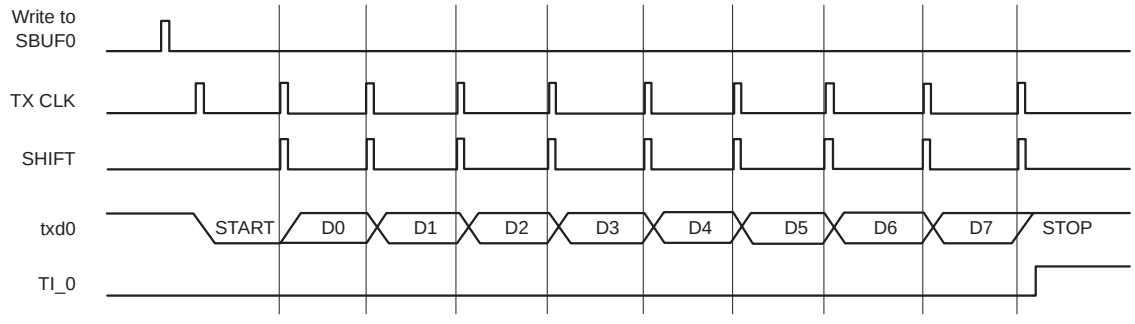


Figure 12−4. Asynchronous 10-Bit Receive Timing



12.7 Modes 2 and 3—11-Bit Asynchronous

Modes 2 and 3 are similar in principle to mode 1, except the data field is extended to nine bits. During reception, nine bits are shifted into an input shift register, of which eight bits are then loaded into the received SBUF register if:

1) RI is 0, and

2) SM2 is 1 and the ninth bit is 1, or SM2 is 0 (that is, the state of the ninth bit does not matter).

If the conditions are not met, the received data is lost, RB8 is not loaded, and RI is not set. If SBUF is loaded, the state of the ninth data bit is copied into RB8 at SCON.2, and RI is set.

Transmission is triggered by a write to SBUF and results in an 11-bit frame consisting of a low-level start bit, eight data bits from SBUF, TB8 from SCON.3, and a high-level stop bit. The start bit begins at the next rollover of the local divide-by-16 counter, and TI is set at the beginning of the stop bit.

For mode 2, the baud rate is $f_{CLK}/64$ if SMOD is 0 (default), or $f_{CLK}/32$ if SMOD is 1. SMOD0 is bit 7 of PCON at 87h and SMOD1 is bit 7 of EICON at D8h.

For mode 3, the baud rate is determined by the overflow rate of Timer/Counters 1 or 2 for serial port 0, or Timer/Counter 1 for serial port 1.

Figure 12−5. Asynchronous 11-Bit Receive

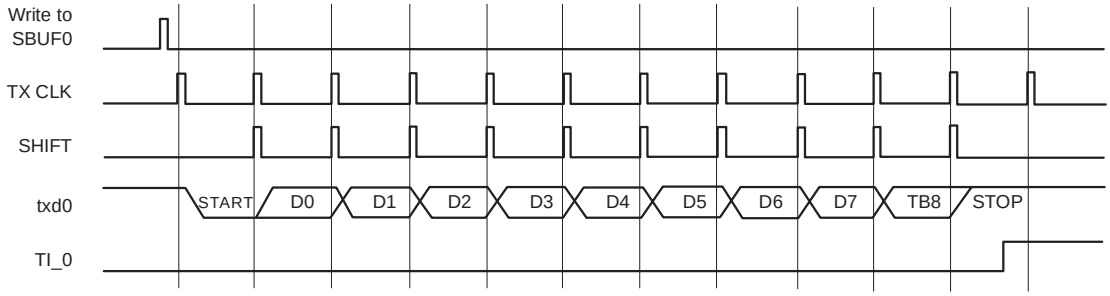
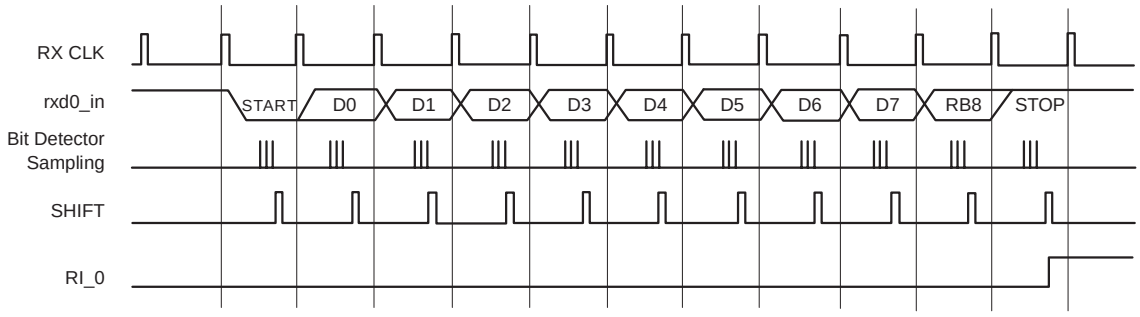


Figure 12−6. Asynchronous 11-Bit Transmit



12.8 Multiprocessor Communications

For serial ports operating in modes 2 or 3 with control bit SM2 = 1, the RI flag will only be set if the ninth bit of a received data field is 1. In this way, a byte may cause an interrupt only when the ninth data bit is 1.

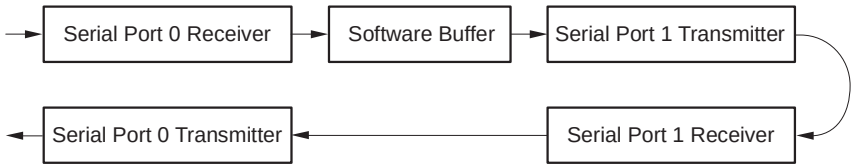
In a multiprocessor system, when a master chooses to send a block of data to one of several slaves, it first transmits an address with the ninth data bit (from SCON.3) at 1. Assuming all slaves initially have SM2 set, then each will be interrupted because RI is set, but only the one matching the address will change its SM2 bit to 0. Thereafter, data bytes with the ninth data bit at 0 will be ignored by unaddressed slaves, but cause an interrupt in the addressed slave.

12.9 Example Program

In Example 12−1, the program has both serial ports operating in mode 1, where asynchronous 8-bit data is preceded by a start bit and succeeded by a stop bit. Since both ports have SM2 = 1 (SCONx.5 = 1), the RI flags are set only if a valid stop bit is received.

Serial port 0 is configured to receive and transmit characters at 9600 baud using Timer/Counter 2, whereas serial port 1 has a non-standard rate of approximately 21 baud using Timer1. Characters received on serial port 0 are buffered in software and presented for transmission via serial port 1. It is assumed that serial port 1 transmitter is looped back to serial port 1 receiver. Characters received at serial port 1 are copied back to serial port 0, as shown in Figure 12−7.

Figure 12−7. Serial Port with Software Buffer



When implemented in a software development environment together with an MSC1210EVM, the user is able to type characters at the PC keyboard and see the same characters on the download window, but with a noticeable delay. The yellow LED will flicker as characters are passed at 21 baud on pin P1.2.

The baud rate of serial port 1 is slow, so the buffer may quickly fill up if keys are typed too rapidly. Impending overflow is indicated by the red LED.

Example 12−1. Serial Port with Software Buffer Code

```
// File Serial01buf.c - Using serial ports 0 and 1 with a buffer
// MSC1210 EVM Switches 1:On  SW3-12345678 SW6-12345678
//                      0:Off     11110111     11110000
#include <Reg1210.h>
#define xtal 11059200
#define BAUD 9600
#define limit 8
sbit RedLed    = P3^4;  // RED LED on EVM
sbit YellowLed = P3^5;  // Yellow LED on EVM
// Join J4 pins 2 and 3 on EVM for loopback

void main(void)
{ data  unsigned char i=limit, j=limit,n=0;   // empty buffer
  idata unsigned char buffer[limit];
  SCON0=0x70;   // Serial Port 0 mode 1 (10 bit asyn.)
  SCON1=0x72;   // Serial Port 1 mode 1 (10 bit asyn.) TI => empty
  RI_0 = RI_1 = 0;    // clear received flags
  CKCON=0x10;   // Timer 1 at fclk/4
  EICON=0x80;   // SMOD1 = 1
  TCON =0x40;   // TR1 is 1
  TMOD =0x00;   // Timer1 13-bit. Baud = xtal/(16*4*8192)=21.1
  T2CON=0x34;   // Timer 2 is rate generator and enabled
  RCAP2=65536-xtal/32/BAUD;
  while(1){
    if (RI_0) {             // wait for key press
      if (n<limit){         // put character into buffer
        if (++i>limit) i=0; // wrap 'on' pointer
        buffer[i]=SBUF0;    // save save character
        n++;                // increment count
        }
      if (n>(limit-2)) RedLed=0; // show impending overflow
      RI_0=0;                    // remove receive flag
      }
    if (TI_1) {             // Is serial Port 2 Tx empty ?
      if (n) {              // Is buffer not empty ?
        if (++j>limit) j=0; // wrap 'off' pointer
        TI_1=0;             // indicate serial port 1 Tx is busy
        SBUF1=buffer[j];    // send character
        n--;                // decrenemt count
        }
```

Example 12−1. (Continued)

```
      else RedLed=1;        // turn off overflow LED
      }
    if (RI_1) {             // is serial port 1 Rx full ?
      SBUF0=SBUF1;          // copy Rx port 1 to Tx port 0
      RI_1=0;
      }
    YellowLed=!RXD1;        // monitor serial port 1 bit stream
    }
}
```

Chapter 13

Interrupts

This chapter describes the MSC121x interrupts.

Topic **Page**

13.1 Description 13-2
13.2 Standard and Extended Interrupts 13-2
13.3 Auxiliary Interrupt Sources 13-4
13.4 Multiple Interrupts 13-5
13.5 Example of Multiple and Nested Interrupts 13-6
13.6 Example of Wake Up From Idle 13-9

13.1 Description

The MSC121x extends the interrupt sources provided by the 8051 architecture in two ways. First, the MSC121x has more interrupts, which have programmable priorities of low or high. Second, the MSC121x has a new group of auxiliary interrupts of highest priority.

When an interrupt occurs, the normal execution of machine-level codes is altered by the forced insertion of an LCALL instruction to an address that depends upon the source of the interrupt. The interrupt itself is generated when the following conditions are present:

1) An asynchronous event sets an interrupt flag.

2) The corresponding interrupt enable bit is set.

3) The group enable bit is set.

4) An interrupt of equal or higher priority has not already occurred.

Normal subroutines are entered via LCALL (or ACALL) instructions, which automatically pushes the Program Counter (PC) onto the stack, and are terminated by the RET instruction, which recovers the PC from the stack.

Interrupt service routines (ISR) are similar but must be terminated by the RETI instruction, which not only recovers the PC from the stack but also restores the interrupt level. Typically, at the start of an ISR, the Program Status Word (PSW) and Accumulator are PUSHed onto the stack and POPed off just before the RETI instruction. Once an RETI instruction has returned control to an interrupted environment and restored the interrupt level, at least one instruction will be executed before another interrupt is acknowledged.

When application code is written in C, protection of the operating context is usually managed by the compiler.

13.2 Standard and Extended Interrupts

Table 13−1 shows the standard and extended interrupts with low or high group priorities and high or low relative priorities. Global Enable = EA (IE.7), where IE is at A8h. Note that in the first column of Table 13−1, the normal text describes the event, while the *italic text* describes how to clear it.

By default, all standard and extended interrupts are grouped with a low priority. However, individual interrupts may be changed to have a high priority by writing ‘1’ to the appropriate bit within either the IP or EIP registers. All interrupts in a high priority group are serviced before those of a low priority group, and those within a group are serviced in the order of relative priority shown in Table 13−1. For example, if Timer 0 and Serial Port 0 are both low priority, then the timer will be serviced before the port. However, if bit 4 of IP at B8h is set to ‘1’, the interrupt from Serial Port 0 will have a higher priority and be serviced before Timer 0.

Any interrupt flag associated with a low or high priority interrupt may be set in software to cause an interrupt, if enabled.

Table 13−1. Standard and Extended Interrupts

Event *Cleared by*	Flag		Enable		ISR Addr	Priority 0 = Low; 1 = High		Relative Priority
	Name	Bit(1)	Name	Bit(1)	00XXh	Name	Bit(1)	
External Interrupt 0 *Notes 2, 3*	IE0	TCON.1	EX0	IE.0	03	PX0	IP.0	High 1
Timer 0 Overflow *Cleared automatically*	TF0	TCON.5	ET0	IE.1	0B	PT0	IP.1	2
External Interrupt 1 *Notes 2, 3*	IE1	TCON.3	EX1	IE.2	13	PX1	IP.2	3
Timer 1 Overflow *Cleared automatically*	TF1	TCON.7	ET1	IE.3	1B	PT1	IP.3	4
Serial Port 0 *Clear RI_0*	RI_0	SCON0.0	ES0	IE.4	23	PS0	IP.4	5
Serial Port 0 *Clear TI_0*	TI_0	SCON0.1	ES0	IE.4	23	PS0	IP.4	5
Timer 2 Overflow *Clear TF2*	TF2	T2CON.7	ET2	IE.5	2B	PT2	IP.5	6
Serial Port 1 *Clear RI_1*	RI_1	SCON1.0	ES1	IE.6	3B	PS1	IP.6	7
Serial Port 1 *Clear TI_1*	TI_1	SCON1.1	ES1	IE.6	3B	PS1	IP.6	7
External Interrupt 2 Positive Edge *Clear IE2*	IE2	EXIF.4	EX2	EIE.0	43	PX2	EIP.0	8
External Interrupt 3 Negative Edge *Clear IE3*	IE3	EXIF.5	EX3	EIE.1	4B	PX3	EIP.1	9
External Interrupt 4 Positive Edge *Clear IE4*	IE4	EXIF.6	EX4	EIE.2	53	PX4	EIP.2	10
External Interrupt 5 Negative Edge *Clear IE5*	IE5	EXIF.7	EX5	EIE.3	5B	PX5	EIP.3	11
Watchdog(3)(4) *Clear WDTI*	WDTI	EICON.3	EWDI	EIE.4	63	PWDI	EIP.4	12 Low

1) Interrupt Enable (IE) is at A8h; Interrupt Priority (IP) is at B8h. Extended Interrupt Enable (EIE) is at E8h; Extended Interrupt Priority (EIP) is at F8h; External Interrupt Flag (EXIF) is at 91h.

2) If the interrupt was edge triggered, the flag is cleared automatically as the service routine is entered; otherwise, the flag follows the state of the pin.

3) May also cause a wakeup from idle, if enabled.

4) For the Watchdog Timer to generate an interrupt, bit 3 of HCR0 must be cleared; otherwise, a Reset (default) will occur.

13.3 Auxiliary Interrupt Sources

Table 13−2 shows the auxiliary interrupts with highest group priority. Global Enable = EAI (EICON.5), where EICON is at D8h, Auxiliary Interrupt Enable (AIE) is at A6h. All these interrupts set the AI flag (EICON.4), which must be cleared in software in addition to the individual interrupt flags. Setting AI in software generates an auxiliary interrupt, if enabled. Note that in the first column of Table 13−2, the normal text describes the event, while the *italic text* describes how to clear it.

When multiple auxiliary interrupts are enabled, the service routine at 0033h can read the Pending Auxiliary Interrupt (PAI) at A5h to identify the interrupt of greatest relative priority. If PAI returns 0, there is no pending auxiliary interrupt.

Pending (active and enabled) interrupts can be identified by testing the corresponding bits in AISTAT at A7h. This allows the programmer to service auxiliary interrupts with arbitrary and even dynamic relative priorities.

Unlike the Interrupt Enable (IE) register at A8h, which returns the value of enable (mask) bits when read, the Auxiliary Interrupt Enable (AIE) register returns the status of interrupt flags before masking. This means that read/modify/write operations on AIE may unintentionally enable interrupts and should not be used.

Unlike flags in the low and high priority groups, no interrupt flag in the highest priority group may be set in software to cause an interrupt. However, AI (EICON.4) can be set to trigger an auxiliary interrupt but a user-specific mechanism must be used to recognize this as a separate source.

For a particular interrupt flag to be set, the corresponding subsystem must be powered up as determined by bits in PDCON at F1h. For example, PDCON.3 must be 0 for an ADC interrupt to occur.

Table 13−2.Auxiliary Interrupts with Highest Group Priority

Event ***Cleared By***	Flag		Enable		ISR Addr	Priority	Relative Priority and Value from PAI
	Name	**Bit**	**Name**	**Bit**	**00XXh**		
DV_{DD} Low-Voltage *Voltage is restored*	EDLVB	AIE.0	EDLVB	AIE.0	33	Highest	1
HW Breakpoint *Set BPCON.7=1*	BP	BPCON.7	EBP	BPCON.0	33	Highest	1
AV_{DD} Low Voltage *Voltage is restored*	EALV	AIE.1	EALV	AIE.1	33	Highest	2
SPI (or I^2C) Receive *Read SPIDATA at 9Bh*	ESPIR	AIE.2	ESPIR	AIE.2	33	Highest	3
SPI (or I^2C) Transmit *Write SPIDATA at 9Bh*	ESPIT	AIE.3	ESPIT	AIE.3	33	Highest	4
Milliseconds Timer *Read MSINT at FAh*	EMSEC	AIE.4	EMSEC	AIE.4	33	Highest	5
ADC Conversion *Read ADRESL at D9h*	EADC	AIE.5	EADC	AIE.5	33	Highest	6
Summation Register *Read SUMR0 at E2h*	ESUM	AIE.6	ESUM	AIE.6	33	Highest	7
Seconds timer *Read SECINT at F9h*	ESEC	AIE.7	ESEC	AIE.7	33	Highest	8

13.4 Multiple Interrupts

In some applications, there may be no interrupts, while in others there may be many. When there is just one interrupt, the service routine is usually relatively easy to write and the model of the timing is simple. However, with multiple sources of different priorities, the complexity in terms of timing and exact behavior grows quickly. Since there are three groups of priority (classed as low, high, and highest), it is possible to have three nested levels of interrupt. For example, the main program may be interrupted by an event of low priority, but the service routine may be interrupted by an event of high priority, which in turn could be interrupted by an event of highest priority.

It is essential that there is no unintentional interaction between different interrupts, and that the operating environment or context is restored prior to termination of an ISR. For all but the simplest of ISRs, it is necessary to save and restore the primary context (PSW and Accumulator) to and from the stack. Similarly, working registers R0 to R7 may need to be PUSHed and POPed, but this is time consuming and can be avoided by register bank switching.

Once the primary context has been PUSHed onto the stack, the value of bits 4 and 3 in the PSW may be changed to select a different bank of 8-bit working registers. In this way, the values in the previous bank are not changed by instructions that reference registers relative to the new bank. It is practical to allocate bank 0 to the main program, bank 1 to low interrupts, bank 2 to high, and bank 3 to highest. Since working registers are also mapped to memory locations, it is possible to modify (and corrupt) any register by writing to an explicit location. For example, R4 of bank 2 is at data address 14h. Care may be needed in this regard when using multiple interrupts.

13.5 Example of Multiple and Nested Interrupts

The example shows interrupt service routines for interrupts of low, high, and highest priority. Based on a clock of 11.0592MHz, the program does the following:

1) Toggles signal sync3 (P1.3) as frequently as possible, subject to servicing interrupts. Assuming the main program is implemented as the instruction CPL P1.3, sync3 will toggle every 0.723µs.

2) Transmits a digit between 0 and 3 at 9600 baud on serial port 0 every 20ms, as triggered by the milliseconds system timer.

3) Uses the interrupt from Timer 0 to toggle sync0 (P1.0) every 278ms.

4) Uses the interrupt from Timer 2 to toggle sync2 (P1.2) every 10ms.

5) Receives characters from serial port 0 via an interrupt and toggles sync1 (P1.1).

6) Shows the level of interrupt nesting by the value of the digit transmitted.

If the time to execute the ISR associated with Timer 0 is short, then most interrupts will occur with respect to the main program. However, every application with multiple interrupts should cater to the least likely combination of events. In this case, it is possible that the main program is interrupted by an overflow from either Timer 0 or Timer 2, which is then interrupted due to a character received on serial port 0, which itself is interrupted by the milliseconds timer. The variable called *level* will then be 3 and cause a ‘3’ to be transmitted because the MsecInt ISR forces a transmit interrupt for serial port 0 by setting TI_0.

The characters most often transmitted are ‘1’ and ‘2’. However, a ‘3’ may be seen occasionally, depending upon the relative timing of the received character with respect to the other interrupts. The probability is affected considerably by the rate at which characters are received, the value of LIMIT, and the efficiency of the code produced by the compiler.

Example 13−1. Multiple and Nested Interrupts

```
// File Interrupts_4.c
// MSC1210 EVM Switches 1:On  SW3-12345678 SW6-12345678
//                      0:Off     11110111     11110000
#include <Reg1210.h>
#include <stdio.h>
#define xtal 11059200
#define BAUD 9600
#define LIMIT 150
#define RATE 100
sbit RedLed    = P3^4;       // RED LED on EVM
sbit YellowLed = P3^5;       // Yellow LED on EVM
sbit sync0     = P1^0;       // Port 1 bit 0
```

Example 13−1. (Continued)

```
sbit sync1     = P1^1;       // Port 1 bit 1
sbit sync2     = P1^2;       // Port 1 bit 2
sbit sync3     = P1^3;       // Port 1 bit 3

data unsigned int j; char level=0, send;

void process(void)
{ data char i;               // simulate additional execution time
  for(i=0;i<LIMIT;i++);
}

void MsecInt(void) interrupt 6 using 3
{ data char temp;
  level++;
  temp=MSINT;                // read MSINT to remove interrupt
  AI=0;                      // remove auxiliary flag
  send='0'+level;            // characters '1' to '3'
  TI_0=1;                    // trigger serial output
  level--;
}

void Serial0Int(void) interrupt 4 using 2
{ level++;
  RedLed=YellowLed;          // monitor
  if(RI_0) {
    sync1=!sync1;            // monitor
    process();               // simulate additional execution time
    RI_0=0;                  // remove Rx flag
    }
  if(TI_0) {                 // test transmit interrupt flag
    TI_0=0;                  // remove Tx flag
    if(send) {SBUF0=send; send=0;}
    }
  level--;
}

void Timer0Int(void) interrupt 1 using 1
{ level++;
  sync0=!sync0;              // monitor
  YellowLed=0;
  process();                 // simulate additional execution time
  YellowLed=1;
  level--;
}
```

Example 13−1. (Continued)

```
void Timer2Int(void) interrupt 5 using 1
{ level++;
  sync2=!sync2;              // monitor
  TF2=0;                     // remove Timer 2 overflow flag
  level--;
}

void main(void)
{ PDCON=0x7D;                // System Timer enabled
  SCON0=0x50;                // Serial 0 enable; RI_0 cleared
  CKCON=0x20;                // Timer 2 at fclk/4; Timers 0 and 1 at fclk/12
  PCON =0x30;                // SMOD = 0 => normal Baud rate eqution
  TMOD =0x22;                // Timers 1 and 0 Auto reload
  TCON =0x50;                // TR1 and TR0 are 1
  TH1  =256-xtal/32/12/BAUD; // Timer 1 reload value
  TH0  =0;                   // Overflows every 256 * 12 * tclk
  T2CON=0x04;                // Timer 2 is auto-reload and TR2 is 1
  RCAP2=65536-xtal/4/RATE;
  MSEC=xtal/1000 - 1;     // 1 ms reference
  MSINT=20 - 1;           // 20 ms interrupt interval
  IP   =0x90;             // Priorities Timer2 'low', Serial0 'high', Timer0 'low'
  IE   =0xB2;             // EA ET2, ES0 and ET0 enabled
  AIE  =0x10;             // EMSEC enabled
  EICON=0x60;             // Auxiliary interrupts enabled
  while(1){
    sync3=!sync3;         // foreground program
    }
}
```

Interrupts from Timers 0 and 2 are both in the low priority group, and are therefore mutually exclusive and share register bank 1. The priority of Serial Port 0 is raised to high by writing a 1 to bit 4 of register IP, and therefore uses a different register bank. Similarly, since the milliseconds interrupt is in the highest group, the service routine is allocated its own register bank.

If interrupts from Timer 0 and Timer 2 are pending at the same moment, Timer 0 will be serviced first because it has a higher relative priority within the low group.

In this particular example, individual service routines may not use registers depending upon the efficiency and optimization level of the compiler. However, the allocation of register banks ensures mutually exclusive contexts and is the usual practice.

The interrupt number used in C is given by (ISR Address − 3) divided by 8.

13.6 Example of Wake Up from Idle

In order to reduce operating power, the MSC121x can be placed into an idle state by writing ‘1’ to bit 0 of PCON at 87h. In this state, the CPU, Timers 0, 1, and 2, and USARTs are not clocked, although other peripherals remain active (unless previously powered-down via bits in PDCON at F1h). Once in the idle state, normal operation is resumed by an enabled auxiliary interrupt or an enabled wake up condition.

Three wake up conditions are enabled by bits in the Enable Wake Up (EWU) SFR at C6h, as shown in Table 13−3.

Table 13−3.EWU—Enable Wake Up

EWU		SFR C6h Reset Value = 00h
Bit #	**Name**	**Action or Interpretation**
7−3		Undefined
2	EWUWDT	Enable wake up on watchdog timer. 0: Disable wake up on watchdog timer interrupt 1: Enable wake up on watchdog timer interrupt
1	EWUEX1	Enable wake up on external 1. 0: Disable wake up on external interrupt source 1 1: Enable wake up on external interrupt source 1
0	EWUEX0	Enable wake up on external 0. 0: Disable wake up on external interrupt source 0 1: Enable wake up on external interrupt source 0

It is possible to synchronize the activity of a program to an external input by repeatedly reading its level. However, this requires more power than configuring an interrupt and placing the MSC1211 into an idle state.

Example 13−2. Wake Up From Idle

```
// File Idle.c
// MSC1210 EVM Switches 1:On  SW3-12345678 SW6-12345678
//                      0:Off     11110111     11110000
#include <Reg1210.h>

sbit sync0 = P1^0;    // Port 1 bit 0
sbit sync1 = P1^1;    // Port 1 bit 1
sbit sync2 = P1^2;    // Port 1 bit 2

data unsigned char j;

void INT0Int(void) interrupt 0 using 1  // No action
{
sync1=!sync1;         // monitor for interest
}

void main(void)
{ IE   =0x81;         // EA EX0
  EWU  =0x01;         // Enable Wakeup
  IT0  =1;            // Falling edge on INT0
  while(1){
    while(!INT0);     // wait for INT0=1
    sync2=0;
    PCON|=1;          // IDLE
    sync2=1;
    for(j=0;j<30;j++) sync0=!sync0;
    }
}
```

If an external interrupt is configured for falling-edge detection, the IDLE bit must be set when the input is high. Similarly, for rising-edge detection, IDLE must be set when the input is low.